UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

NÚMERO DE CLAN;UN ENFOQUE MOLECULAR

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

M A T E M Á T I C O

P R E S E N T A:

YAZMIN PINEDA AMADOR

DIRECTOR DE TESIS:
DR.RICARDO STRAUSZ SANTIAGO
2011

1. Datos del alumno
   Pineda
   Amador
   Yazmin
   5690-1227
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Matemáticas
   301220356
2. Datos del tutor
   Dr.
   Ricardo
   Strausz
   Santiago
3. Datos del sinodal 1
   Dr.
   Ricardo
   Gómez
   Aiza
4. Datos del sinodal 2
   Dr.
   Sergio
   Rajsbaum
   Gorodezky
5. Datos del sinodal 3
   Dr.
   Juan José
   Montellano
   Ballesteros
6. Datos del sinodal 4
   Dr.
   Javier
   Bracho
   Carpizo
7. Datos del trabajo escrito
   Número de clan; un enfoque molecular
   61pp
   2011

Agradeciminentos

Desde que iba en la primaria me gustaba adelantarme en los libros de matemáticas, y desde que llegue a la secundaria hasta la licenciatura tuve la fortuna de tener profesores de matemáticas que disfrutaban su materia y yo junto con ellos, en el CCH.Ote, me obsesionaban las mates porque cuando el profesor solo dejaba los ejercicios impares yo llegaba con casi todos resueltos (pares e impares). Y ya en la facultad de ciencias no me quitaban el sueño, pero soñaba con ellas, una vez durmiendo en el camión en mi sueño trataba de calcular la hora con números complejos (jajaja). Conclusión no soy una genio en las matemáticas pero siempre me seducen y me divierten.

Por todo esto y porque en mi casa siempre me dijeron tienes que estudiar algo que te guste y disfrutes. Quiero dar gracias a Dios y a cada uno de mis familiares que con su apoyo y ejemplo no hubiera tenido el coraje suficiente para concluir mis estudios. En especial a mi madre incansable,, que desde el Kínder hasta la Universidad se levantó tempranito para irme a dejar.

A mis amigos doy las gracias por su paciencia y compañía. Entre ellos a Rosa y Mauricio mis primeros amigos de la facultad que me hicieron dar mis primeras carcajadas en la biblioteca. A Daniel, David y Leonardo por sus críticas y tardes de café acompañadas de una buena tarea de cálculo. A Karina por su ejemplo de método de estudio y libros. Abel por tu apoyo estando cerca y lejos. Miriam y Ale por su ejemplo y dedicación, Juanchito, Mario y Vladi por compartir cosas padres y materias difíciles. Mike por creer y depositar toda tu confianza, admiración y cariño en mí. Y gracias a todos los que hicieron posible este trabajo, entre ellos a Lucy por tus consejos de Latex, Memo y Rosal por las impresiones, internet, ayuda técnica y , Adrián por ayudarme con la búsqueda bibliográfica y permitirme trabajar contigo. Gracias a mis sinodales por su tiempo, paciencia y críticas.

A los buenos profesores que tuve en la carrera, en especial a Ricardo Strausz que me alentó y motivo a seguir adelante cuando quería abandonar la carrera y que por azares del destino termino como mi asesor, dando como resultado esta bonita tesis.

Índice general

Capítulo 1

Introducción

El *problema del clan máximo* (PCM) consiste en encontrar en una subgráfica, un conjunto maximal de vértices que están completamente unidos por aristas. Este problema tiene aplicaciones en: teoría de códigos, diagnóstico de errores, visión computacional, análisis de agrupamiento, recuperación de información, aprendizaje automático, minería de datos, entre otras. (cf. [7]) Richard Karp demostró en 1972 que el problema de decisión asociado a PCM es un problema *NP-completo*; es decir, no se sabe si tiene un algoritmo que lo resuelva en tiempo polinomial de forma determinística (con una máquina de Turing). El primer algoritmo de fuerza bruta que se ocurre es checar todas las subgráficas inducidas, que serían 2^n donde n es el número de vértices.

En este trabajo se revisarán y comparán algunos de los algoritmos, que resuelven nuestro problema en tiempo polinomial utilizando cómputo molecular. Tienen en común el uso de moléculas de *Ácido Desoxirribonucleico* (ADN), un novedoso tipo de cómputo molecular que tiene como ventajas: rapidez al realizar muchas operaciones en paralelo, bajo consumo de energía, y densidad en el almacenamiento de la información.

El cómputo con ADN fue implementado por primera vez por Leonard Adleman (1994) [1] quien resolvió el problema de la trayectoria hamiltoniana. La primera implementación para PCM fue hecha por Ouyang et al. (1997) [6]. Ellos proponen un algoritmo donde a cada posible clan de una gráfica se le asigna una cadena doble de ADN, y base en la gráfica complementaria se van cortando las cadenas, para ir descartando los clanes que no pertenecen a la gráfica. Como resultado se obtienen todas las cadenas que

representan clanes en la gráfica, para luego poder ser medidas y secuenciadas, proporcionando el subconjunto exacto de vértices que forman un clan máximo.

A partir de que fue presentado el trabajo de Ouyang, otros trataron el mismo problema desde diferentes puntos de vista. Entre ellos Head et al.(1999) [3] con la ventaja de no crear todos los posibles clanes de una gráfica; es decir, no tiene que construir muchas cadenas distintas de ADN asociadas a los posibles clanes en la gráfica. Comienza con una sola molécula de ADN en forma circular, que se modifica conforme el algoritmo avanza y produce las distintas cadenas que representan todos los clanes pertenecientes a la gráfica. Propone un algoritmo para encontrar el máximo conjunto independiente en la gráfica complementaria, que es equivalente al problema del clan máximo.

Bäck et al (1999) [2], con el propósito de mejorar el algoritmo propuesto por Ouyang, en el sentido de que funcione para gráficas con mayor número de vértices, relaciona cálculo evolutivo y cálculo con ADN, para que la solución evolucione en lugar de ser extraída de un conjunto grande de posibles soluciones, evitando también crear todas las posibles soluciones desde el principio.

McCaskill (2001) [5] propone un sencillo algoritmo para seleccionar todos los clanes que pertenecen a la gráfica. Trabajando con cadenas sencillas de ADN (no dobles), implementa cálculo con ADN dentro de microreactores con microflujo, donde tienen lugar las operaciones que se realizan en el laboratorio, programandolo ópticamente.

Zimmermann (2002) [9] utilizando un modelo de etiquetado donde las cadenas de ADN son parcialmente dobles, propone un algoritmo que encuentra todas las posibles subgráficas inducidas por $\binom{k}{2}$ aristas. De éstas se escogen las que tengan k vértices. Para obtener todos los clanes de tamaño k, el algoritmo también reporta si no hay un clan de ese tamaño en nuestra gráfica.

Por último, Marc García-Arnau et al (2007) [4] utilizan un sistema de membranas anidadas (P-sistema), donde cada membrana tiene sus reglas; las cadenas de ADN representando clanes son construidas gradualmente conforme van pasando por cada membrana. Al salir de la última membrana resultan todos los clanes contenidos en la gráfica de los cuales hay que escoger

el más grande.

1.1. Problema del Clan Máximo (PCM)

Para entender el Problema del Clan Máximo y los tipos de cálculo molecular que lo resuelven, es necesario dar algunos conceptos básicos de la teoría de gráficas; y por ser un problema NP-completo se verá un poco de complejidad computacional. Por otro lado, se explicarán términos biológicos, técnicas y operaciones aplicadas en la implementación para la primera aproximación molecular para PCM. Algunas de estas técnicas también se utilizarán en las siguientes aproximaciones en el capítulo 3, donde cada aproximación tendrá sus propios conceptos con nuevas técnicas y operaciones biológicas, explicadas al principio de cada sección.

1.1.1. Teoría de gráficas

Una *gráfica* es una pareja de conjuntos $G = (V, A)$, donde V es un conjunto de puntos llamados *vértices* y A es un conjunto de *aristas* que unen algunos pares de vértices. Una *subgráfica* $H = (V', A')$ de G es una gráfica tal que $V' \subseteq V$ y $A' \subseteq A$. Un *clan*, es un subconjunto de vértices $V' \subseteq V$ tal que cualesquiera dos elementos en V', están conectados por una arista en A. Entonces el *problema del clan máximo* es: dada una gráfica cualquiera, encontrar la subgráfica con mayor cantidad de vértices, conectados todos con todos por sus aristas; es decir, una *gráfica completa,* denotada como K_n, donde n es el número de vértices de la gráfica.

Obsérvese que este problema es equivalente a encontrar el máximo *conjunto independiente* aquel que entre cualesquiera dos vértices no están unidos por una arista. La *gráfica complementaria*, denotada por $\bar{G}$, es la gráfica que contiene los mismos vértices que G y tiene como aristas, aquellas que no están en G. Esta gráfica complementaria dará información importante en la siguiente aproximación, para saber qué vértices no están en un posible clan, ya que al estar la arista en $\bar{G}$, quiere decir que no está en G, y por tanto alguno de los dos vértices incidentes a esta arista, no pueden pertenecer a un clan simultáneamente. En la Figura 1.1 se tiene el ejemplo de una gráfica $H = (V, A)$ con $V = \{a, b, c, d, e\}$ y $A = \{(a, b), (a, e), (b, e), (b, d), (e, d), (d, c)\}$ y tiene como clan máximo al conjunto de vértices $\{a, b, d, e\}$ formando la sub-

gráfica K_4 y del lado derecho su gráfica complementaria $\bar{H}$ quien tiene como máximo conjunto independiente los vértices $\{a, b, d, e\}$.

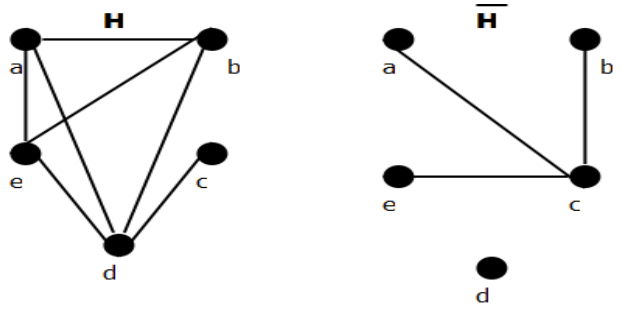

Figura 1.1: Gráfica H y su complementaria $\bar{H}$

1.1.2. Complejidad computacional

La teoría de la complejidad computacional estudia los recursos requeridos para resolver un problema como son el tiempo y el espacio; por su parte la teoría de la computabilidad se interesa en expresar los problemas como algoritmos sin tener en cuenta la información sobre los recursos necesarios para ello. Para abstraer las variaciones entre los diferentes sistemas computacionales se utiliza una máquina de Turing como un refente fijo. Las máquinas de Turing pueden ser pensadas como algoritmos que resuelven problemas expresados con sucesiones de símbolos que representan instancias de cualquier problema bajo algún esquema de codificación.

Las clases de teoría de la complejidad computacional se ha dividido principalmente en dos; las clases P y NP. Un problema de decisión pertenece a la *clase* P si existe un algoritmo determinístico en tiempo polinomial sobre una máquina de Turing que lo resuelva. Un problema pertenece a la *clase* NP si es posible generar una solución con una máquina de Turing no determinística y luego verificarla en tiempo polinomial. Dentro de los problemas NP existen problemas más duros (los más difíciles de la clase NP) conocidos como problemas *NP-completos*; es decir, son los posibles problemas contenidos en *NP-P*, los cuales tienen la propiedad que si alguno de ellos tiene un algoritmo determinístico de tiempo polinomial que lo resuelva, entonces todos los problemas NP podrían ser resueltos en tiempo polinomial por algún algoritmo determinístico.

1.1.3. Técnicas en biología

Los algoritmos que se verán en este trabajo son implementados en el laboratorio y requieren de cadenas de Ácido Desoxirribonucleico (ADN). Éstas están formadas por moléculas llamadas *nucleótidos*, los cuales difieren sólo en un elemento llamado *base*. Hay cuatro bases: citosina, tiamina, guanina y adenina, abreviadas comúnmente como C, T, G y A, respectivamente. La clásica *doble hélice* o también llamada doble cadena (dsADN) está formada por dos cadenas de ADN al unirse. Esta unión ocurre por la atracción de $A-T$ y $G-C$ y son conocidas como pares de bases complementarias.

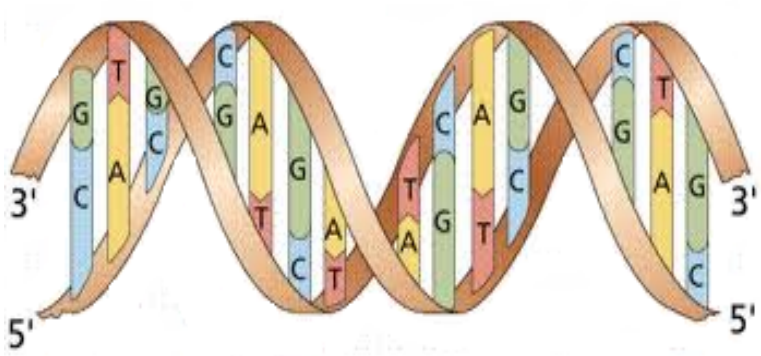

Figura 1.2: Esquema de una doble cadena de ADN

Los *oligonucleótidos* son cadenas cortas de ADN de aproximadamente 20 nucleótidos unidos, en este trabajo también se hará referencia ellos como *oligos*. Las dsADN tienen como unidad de medida los *bp*, pares de bases, por sus siglas en inglés. Para realizar los cálculos con ADN se necesitan aplicar una serie de operaciones biológicas a un conjunto de cadenas, que a continuación se describen.

Desnaturalización. Ocurre en una cadena doble de ADN cuando se puede dividir en dos cadenas sencillas, por medio del calor. Si esta solución se enfría permite que las cadenas complementarias se unan de nuevo. La desnaturalización se produce también variando el pH o a concentraciones salinas elevadas. Si se restablecen las condiciones, el ADN se naturaliza y ambas cadenas se unen de nuevo.

Figura 1.3: Naturalización y desnaturalización de ADN

Si una cadena sencilla contiene una discontinuidad, ésta puede ser reparada por una enzima llamada ADN ligasa. Esto permite crear una cadena unificada de varias cadenas junto con sus respectivos complementos. Por ejemplo en la Figura 1.4 se representan tres cadenas sencillas diferentes, donde las dos cadenas mas cortas tienen una discontinuidad. Éstas pueden ser reparadas por ADN ligasa para después formar una cadena doble.

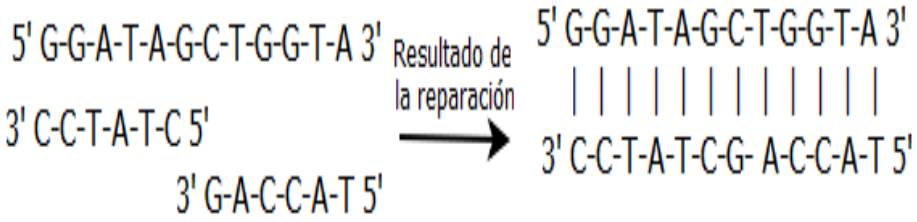

Figura 1.4: Ligación

Otra técnica importante que se usará es *electroforesis en gel*, que sirve para clasificar las cadenas de ADN por tamaños. Electroforesis es una técnica que ayuda en el estudio del movimiento de las biomoléculas con una carga neta a través de un campo eléctrico. Su migración depende de la forma, tamaño, carga y composición química; por ejemplo, si se tiene una mezcla, cada molécula presentará una carga y tamaño único, por lo tanto la movilidad y velocidad de migración en el campo eléctrico para cada molécula es única y se separan en bandas. Una vez que las moléculas de ADN tienen una carga negativa, cuando se colocan en un campo eléctrico tienden a moverse hacia el polo positivo. Una representación simplificada de electroforesis en gel se muestra en la Figura 1.5.

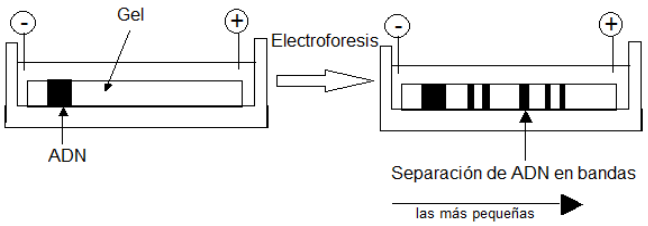

Figura 1.5: Proceso de electroforesis en gel

La visualización de los resultados, se logra mediante la tinción del ADN con tinte fluorescente, y a continuación se observa el gel bajo la luz ultravioleta. En esta etapa el gel se suele fotografiar como lo muestra la Figura 1.6 y se interpreta de la siguiente manera; cada línea corresponde a una muestra de ADN, la línea 1 es conocida como *línea de registro*, y contiene varios fragmentos de ADN de longitud conocida. Se agrupan en forma de banda horizontal los fragmentos de ADN de la misma longitud. Los fragmentos mas grandes se quedan en la parte de arriba y los más chicos en la parte inferior. El brillo de una banda en particular depende de la cantidad de ADN presente en la muestra.

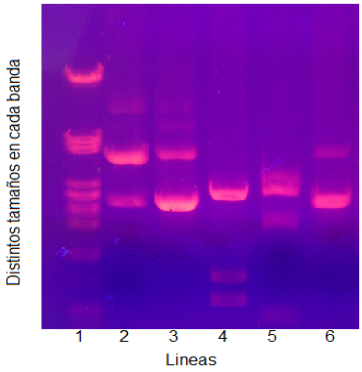

Figura 1.6: Fotografía de electroforesis en gel

Reacción en cadena de la polimerasa (PCR), es una técnica de biología molecular para obtener un gran número de copias de un fragmento de ADN en particular (*amplificación*). En resumen esta técnica consiste en hacer cambios de temperatura llamados ciclos, y sus principales pasos son: dada una dsADN el primer proceso que ocurre para duplicarse es la *desnaturalización*,

la separación de las dos cadenas complementarias; así, cada una de ellas sirve como molde. Este proceso se lleva a cabo por un aumento de temperatura a la mezcla de reacción (92-98^0 C, de $30 - 90$ segundos). El segundo paso es *alineamiento* y consiste en que los *cebadores* (secuencias cortas de nucleótidos a partir de la cual la ADN polimerasa inicia la síntesis de una molécula nueva de ADN) hibridan y renaturalizan al ADN molde. Generalmente se utilizan dos cebadores distintos, cada uno de ellos tiene secuencia complementaria a una de las dos cadenas de ADN molde. Los cebadores se alinean con sus extremos 3'. En este paso, la mezcla de reacción se enfría. Por último, el tercer paso se le agrega a la mezcla de reacción la enzima ADN polimerasa; esta extiende los cebadores en dirección $5' - 3'$, utilizando como molde al ADN de cadena sencilla unido al cebador. El producto son dos moléculas de ADN de doble cadena con los cebadores incorporados en el producto final. Cada grupo de tres pasos se denomina *ciclo termal*. Empezando con una molécula de ADN, el primer ciclo produce dos moléculas de ADN, dos ciclos producen cuatro, tres producen ocho etc. (ver la Figura 1.7)

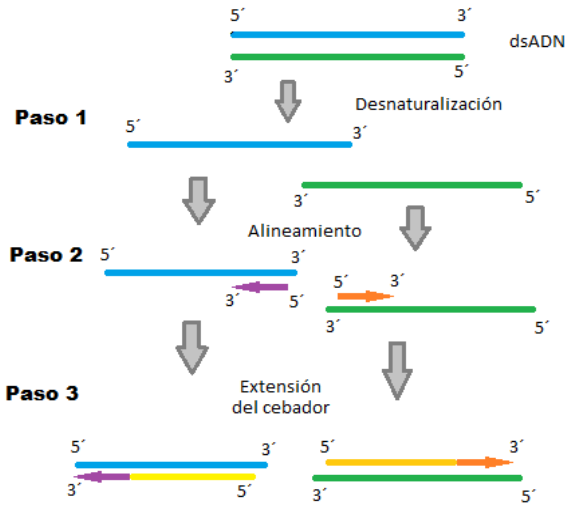

Figura 1.7: Los tres paso principales del PCR

Las proteínas que actúan como *enzimas* aceleran la mayoría de las reacciones químicas, por ejemplo; la enzima de restricción ligasa, que acelera la ligación de cadenas de ADN. Pero también existen las *enzimas de restricción* que reconocen una secuencia específica dentro de una cadena de ADN

conocido como *sitio de restricción*. Cualquier cadena de ADN que contenga el sitio de restricción dentro de sus secuencias es cortada por la enzima. Por ejemplo la doble cadena de ADN en la Figura 1.8a) es cortada por la enzima de restricción RsaI, la cual reconoce el sitio de restricción GTAC. La enzima rompe a la mitad del sitio de restricción. Algunas enzimas como RsaI dejan finales romos, mientras que otras enzimas dejan finales cohesivos. Por ejemplo la cadena doble de ADN en la Figura 1.8b) es cortada por la enzima Sau3AI, la cual reconoce el sitio de restricción GATC.

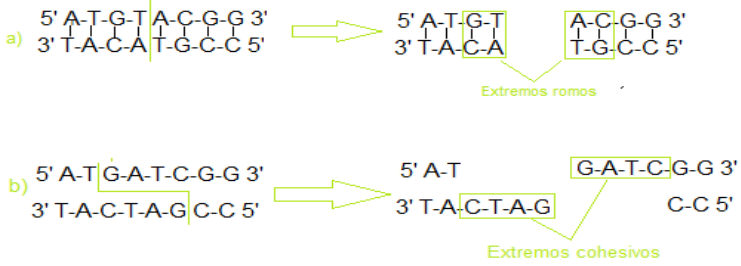

Figura 1.8: Ejemplos de enzimas de restricción

Los algoritmos que utilizan cómputo con ADN actualmente requieren una construcción de los datos iniciales para el problema de gráficas a resolver. Usualmente los datos se representan con sucesiones de números binarios, dando a cada dígito una secuencia de ADN que representa su valor y su posición dentro de la sucesión. Estas secuencias serán nuestro conjunto inicial a ordenar (ver Figura 1.9), en donde se muestran oligonucleótidos tales que puedan reconocer solamente fragmentos adyacentes. El *ensamble en paralelo* (POA Parallel Overlap Assembly por sus siglas en inglés) implica ciclos que son la iteración de unir, reconocer y extender construyendo moléculas largas de fragmentos pequeños de ADN. La cadena de posición se une a una cadena complementaria del siguiente oligo y pueden ser extendidas por una enzima ADN polimerasa, para formar una cadena doble de ADN larga. Una *fuente de datos* es el resultado después de unos cuantos ciclos, da todas las posibles combinaciones de ceros y unos representadas por cadenas dobles de ADN.

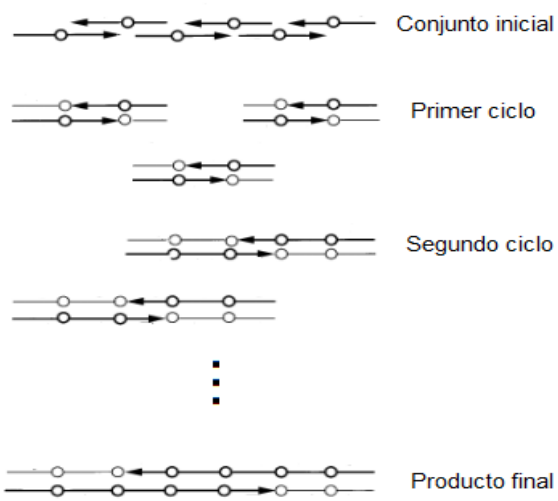

Figura 1.9: POA las flechas representan cadenas sencillas de ADN y las líneas representan la extensión

Capítulo 2

Primera aproximación molecular para PCM

Inspirado por los resultados obtenidos por Adleman en 1994, con los cuales muestra que el ADN puede ser utilizado para resolver problemas computacionalmente difíciles, Ouyang et al. presentan una solución, usando cómputo molecular, para el problema del clan máximo que en esencia es como sigue. Procede con el siguiente algoritmo: dada una gráfica G con n vértices y m aristas, se crean todos los posibles clanes contenidos en la gráfica completa con n vértices (K_n), y con ayuda de la gráfica complementaria $\bar{G}$, se eliminan todos aquellos clanes que **no** están contenidos en G. Así, quedan sólo los clanes que si están contenidos en G de los cuales se escoge el más grande.

2.1. Algoritmo

1 Para una gráfica G con n vértices $\{x_1, x_2, ..., x_n\}$, crear las 2^n sucesiones de números binarios de longitud n, que representan cada posible clan o subgráfica inducida por los vértices en K_n. El 1 en el lugar del i-ésimo dígito, significa que el vértice x_i pertenece a un clan, y el 0 que no pertenece. De esta manera se representa al conjunto de todos los posibles clanes de una gráfica con n vértices. Al conjunto de todos los números binarios de longitud n se llamará *fuente de datos*.

2 Encontrar pares de vértices en la gráfica G que no están conectados

por una arista; es decir, se calcula la gráfica complementaria $\bar{G}$.

3 Eliminar de la fuente de datos todas las sucesiones de números binarios, que contengan conexión en la gráfica complementaria; es decir, si la arista (x_i, x_j) está en $\bar{G}$ (por tanto no está en G), entonces todas las sucesiones, que tengan 1 en el lugar del j-ésimo dígito y en el lugar del i-ésimo dígito no pueden pertenecer a un clan en G. Son eliminadas estas sucesiones de números binarios. El resto de la fuente de datos corresponde a todos los clanes en la gráfica original.

4 En el resto de la fuente de datos encontrar el número binario con el mayor número de 1's. Esto proporciona cuál es número de vértices en G, tal que forman clan máximo.

Justificación del algoritmo

Sea $G = (V, A)$ una gráfica de orden n, en el primer paso del algoritmo se crean todas las subgráficas inducidas por los vértices de K_n, obsérvese que cada una de estas subgráficas es una completa de orden k con $1 \leq k \leq n$. Para el segundo paso, se revisan todas las aristas que no están en G; es decir, se calcula $\bar{G}$. En el paso 3, se eliminan todas las subgráficas inducidas por los vértices de K_n tal que contengan la arista a_i en $A(\bar{G})$ ya que si contienen alguna de éstas, no pueden formar un clan en G (por definición de clan los vértices tienen que estar unidos todos con todos). Al final del proceso de eliminación quedan solo subgráficas de K_n que son clanes en G. Supóngase que alguna de estas subgráficas no es un clan en G entonces existe una subgráfica que le falta al menos una arista quiere decir que esta arista estaba en $\bar{G}$, pero por el proceso de eliminación se debió eliminar esta subgráfica **!**. Por lo tanto todas las subgráficas que quedan después del proceso de eliminación son clanes en G. Para el último paso del algoritmo, se cuentan cuántos vértices tiene cada clan y de todos se toma alguno de tamaño máximo.□

Ejemplo:

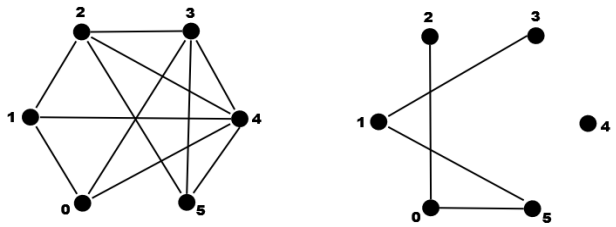

Figura 2.1: Del lado izquierdo se tiene la gráfica H y del lado derecho la gráfica complementaria $\bar{H}$.

Paso 1 Se tiene la gráfica con 6 vértices, $\{0, 1, 2, 3, 4, 5\}$ de la Figura 2.1. Se calcula la fuente de datos de los 64 números binarios de longitud 6, representando todos los posibles clanes que contiene G. Se construyen éstos de la siguiente forma: se escribe comenzando por el lado izquierdo, la ausencia o presencia de un vértice, de tal forma que la posición del i-ésimo dígito le corresponde al vértice $i-1$ de H, asignando 0 si el vértice no pertenece a un clan y 1 en caso de pertenecer a algún clan. En el ejemplo el clan formado por los vértices 1,2,4 en H, está representado por el número binario 011010. A continuación se presentan las $64=2^6$ sucesiones de números binarios que representan todos los clanes posibles contenidos en K_6.

```
000000,110000,001001,001101,100101,001111,110111,011100,
000001,011000,000101,100011,111100,011011,101111,110001,
000010,001100,101000,010011,101001,011101,011111,111101,
000100,000110,100100,001011,101110,011110,111111,100111,
001000,000011,100010,010101,110110,101011,111011,010110,
010000,100001,010010,101010,110011,101101,010111,011001,
100000,010001,010100,101100,110101,111010,110100,000111,
001010,001110,110010,100110,111001,111110,011010,111000,
```

Paso 2 Calculando la gráfica complementaria, se obtienen los pares de vértices que no están conectados en H; como son (1,3), (1,5), (0,2), (0,5).

Paso 3 Se eliminan todas las sucesiones binarias que tengan la siguiente configuración: `X1X1XX,X1XXX1,1X1XXX,1XXXX1,` que corresponden a las

aristas (1,3), (1,5), (0,2), (0,5) en $\bar{H}$, respectivamente. Aquí, `X` puede ser 0 o 1. Por estar en $\bar{H}$ estas aristas no pueden pertenecer a ningún clan en H, y eliminando todos estos números binarios lo que nos quedan son todos los clanes posibles contenidos en H. Estos son los representados por las siguientes sucesiones después de este proceso de eliminación.

```
000000,000100,100100,001100,000110,110010,000111,001101,
100000,000010,100010,001100,000101,001110,001111,100110,
010000,000001,011000,001010,000011,001011,010010,001001,
001000,110000.
```

Paso 4 En H existen 26 clanes representados por las sucesiones de números binarios anteriores. De estas sucesiones es necesario verificar cuál posee más 1's y ésta es `001111` la cual indica el máximo clan en H y que éste corresponde a la gráfica inducida por los vértices {2,3,4,5}.

2.2. Implementación molecular del algoritmo

Como se vio, el primer paso del algoritmo es crear la fuente de datos de tal forma que cada sucesión de números binarios será representada por una doble-cadena de ADN. La pregunta es: ¿Cómo diseñar estas cadenas de dsADN? Primero a cada dígito de un número binario le corresponden dos secuencias de ADN: una que representa cuando el valor es 0 y otra cuando el valor es 1,(V_i^0 y V_i^1 respectivamente) y otras dos secuencias P_i y P_{i+1} indicando la posición del i-ésimo dígito en la sucesión y la posición del siguiente dígito respectivamente. En conjunto nos quedan los siguientes fragmentos de oligonucleótidos: $P_iV_i^0P_{i+1}$, $P_iV_i^1P_{i+1}$ cuando i es par, y $\overline{P_{i+1}V_i^0P_i}$ y $\overline{P_{i+1}V_i^1P_i}$ cuando i es impar, donde la barra nos indica que son secuencias complementarias. Por convención las secuencias se leen de $3'$ a $5'$ y las secuencias complementarias de $5'$ a $3'$. Así, para una molécula de ADN representando un número binario de longitud seis, habrá 6 secciones de valor (V_0 a V_5) intercalados secuencialmente entre siete secciones de posición (P_0 a P_6) (Figura 2.2).

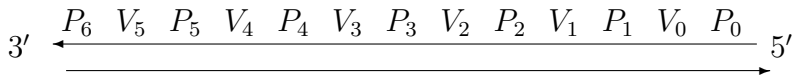

Figura 2.2: Estructura de una sucesión binaria en dsADN

Los P_i y P_{i+1} estarán constituidos por 20 bases, V_i^0 con diez bases y V_i^1 cero bases. Las bases A, C, G, T, en principio fueron asignadas aleatoriamente, pero para evitar apareamientos no deseados, se evitó tener homologías mas grandes de 4 *bp*. Además, cuando la cadena contenga V_i^1 a la mitad de las 40 bases se colocaran ciertas secuencias, llamadas sitios de reconocimiento, que más adelante servirán para que las enzimas de restricción reconozcan el sitio donde deben cortar las cadenas que corresponden a sucesiones que representan clanes no pertenecientes a H.
Ahora, ¿por qué 20 bases y no menos?. Se piden que sean lo bastante chicas para que las cadenas se peguen a 60^o y lo suficientemente grandes para evitar similitud no deseada en las secuencias, y por lo tanto enlaces no deseados entre cadenas. Se eligen 10 bases para V_i^0 para facilitar la distinción de tamaños a la hora de aplicar electroforesis en gel, pero a su vez son suficientemente pequeñas para evitar hibridaciones no deseadas.
La primera y última sección de posición, P_0 y P_6 es necesaria para la amplificación (tienen la función de un cebador) en el momento de aplicar PCR. Con las observaciones anteriores se crearon los siguientes 12 oligonucleótidos: (Tabla 1) dos por cada vértice en H, uno representando que el vértice pertenece a al clan y otro que no pertenece.

En la siguiente tabla se tienen los 12 oligonucleótidos para construir la fuente de datos, donde cada cadena contiene secuencias de posición P_i y secuencias de valor V_i^j, donde j indica el valor de V_i. Las secuencias de valor V_i^0 son escritas con letras minúsculas, y los sitios de reconocimiento para las enzimas de restricción en V_i^1 son las letras subrayadas, estas enzimas son Afl II, Hind III, Spe I, Sph I, Stu I, y Xho I y se aplican para cortar $V_0^1, V_1^1, V_2^1, V_3^1, V_4^1, V_5^1, V_6^1$ respectivamente.

Fragmento de ADN	Secuencia (5' a 3')
$P_0V_0^0P_1$	CGTAGAATTCTGCGAACCTTgacgcggcagAAGAGTCACCTATCAGTAAG
$P_0V_0^1P_1$	CGTAGAATTCTGCGAAC*CTTAAG*AGTCACCTATCAGTAAG
$\overline{P_2V_1^0P_1}$	AGTAATCTCTCTTCACGAAGcggttatgaaCTTACTGATAGGTGACTCTT
$\overline{P_2V_1^1P_1}$	AGTAATCTCTCTTCACG*AAGCTT*ACTGATAGGTGACTCTT
$P_2V_2^0P_3$	CTTCGTGAAGAGAGATTACTccggtcacttAGTGCTAACAAGCCTGCGCA
$P_2V_2^1P_3$	CTTCGTGAAGAGAGATT*ACTAGT*GCTAACAAGCCTGCGCA
$\overline{P_4V_3^0P_3}$	CCTTTATCTCAAAGACTGCAaatcgtcaggTGCGCAGGCTTGTTAGCACT
$\overline{P_4V_3^1P_3}$	CCTTTATCTCAAAGACT*GCATGC*GCAGGCTTGTTAGCACT
$P_4V_4^0P_5$	TGCAGTCTTTGAGATAAAGGaaaaacccacCCTACGTGTGAAATAGACTC
$P_4V_4^1P_5$	TGCAGTCTTTGAGATAA*AGGCCT*ACGTGTGAAATAGACTC
$\overline{P_6V_5^0P_5}$	CCCTGGATCCCGCCCCTCTCagatcggtggGAGTCTATTTCACAGCTAGG
$\overline{P_6V_5^1P_5}$	CCCTGGATCCCGCCCCT*CTCGAG*TCTATTTCACACGTAGG

Ya que se tienen los 12 oligonucleótidos, se aplica ensamble en paralelo (POA) para construir nuestra fuente de datos. Se comienza mezclando los 12 fragmentos de oligonucleótidos para un ciclo termal. Durante cada ciclo, las cadenas de posición reconocen la cadena complementaria del siguiente oligonucleótido. Los finales $3'$ se extienden en la presencia de la polimerasa, para formar dsADN más grandes. Después de unos cuantos ciclos termales se crean todas las combinaciones de $V_0^kV_1^kV_2^kV_3^kV_4^kV_5^k$, $k\in\{0,1\}$ (Figura 2.3). El POA fue seguido por PCR para hacer millones de copias de cada elemento de la fuente de datos teniendo como modelo el resultado del proceso POA. Y con las secuencias P_0 y P_6 actuando como cebadores, sólo aquellas moléculas que tengan éstas secuencias en sus extremos fueron amplificadas exponencialmente. El resultado de POA y PCR son mostrados en las columnas 2 y 3 respectivamente. Así, de acuerdo a la codificación de cada subgráfica de H, la longitud de las cadenas puede variar entre, 140 a 200 *bps* que es lo que se observa en la columna 3. (Figura 2.4).

Ya que se tiene construida la fuente de datos, con la ayuda de la gráfica complementaria $\bar{H}$, se aplica el paso 3 del algoritmo. Incorporando las enzimas de restricción a la fuente de datos, éstas rompen el ADN en los sitios especifícos de reconocimiento, integrados en las secuencias V_i=1. Al romper estas cadenas no son amplificadas exponencialmente por PCR. Entonces sólo se amplificarán aquellas secuencias que no sean cortadas por las enzimas de restricción. Así es como se eliminan las cadenas que representan sucesiones

que a su vez representan clanes no contenidos en H.

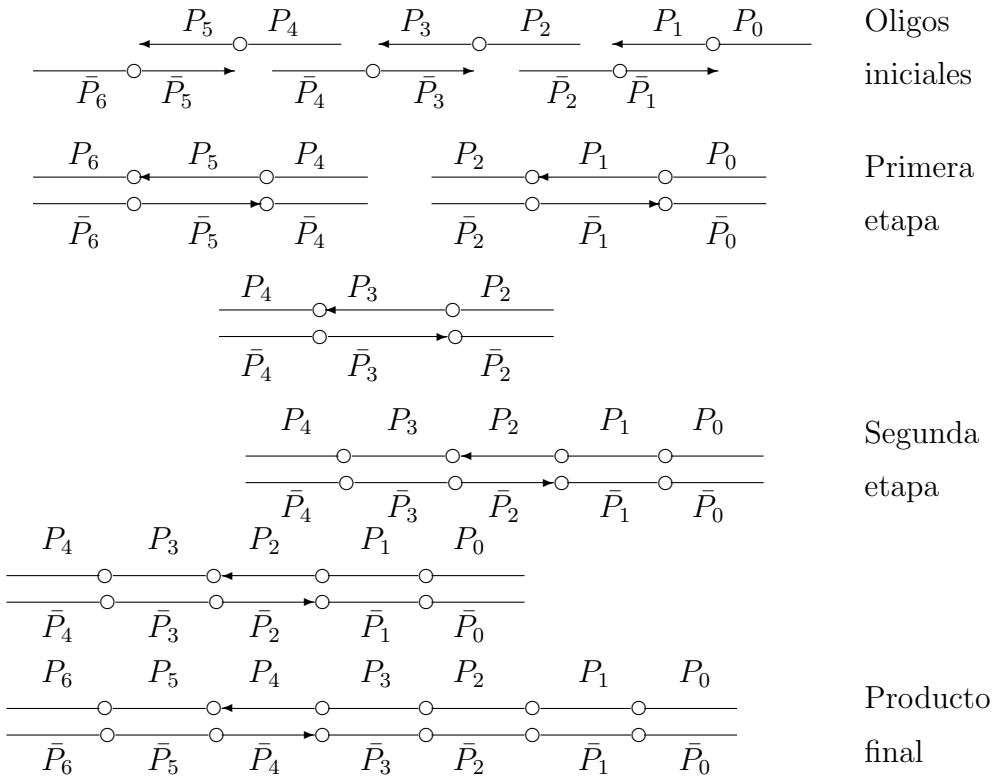

Figura 2.3: ensamble en paralelo (POA)

Por ejemplo, para cortar todas las cadenas que indican que los vértices 0 y 2 están en un clan que pertenece a H, (Figura 2.1), primero se divide la fuente de datos dentro de dos tubos, t_0 y t_1. En t_0 se cortan las cadenas conteniendo V_0^1 con la enzima de restricción Afl II, y en t_1 las cadenas que contienen V_2^1 con la enzima de restricción Spe I. Luego, se combinan los tubos en uno sólo que ya no contiene cadenas de la forma `XXX1X1`. Cuatro operaciones como éstas (con diferentes enzimas de restricción) eliminan todas las cadenas conectadas por aristas en la Figura 2.1, que corresponden a `XXX1X1` (0-2 arista), `1XXX1X` (arista 1-5), `1XXXX1` (arista 0-5), y `XX1X1X` (arista 1-3). Y el resto de cadenas que no fueron cortadas fueron amplificadas por PCR, ya que contenían en sus finales a P_0 y $\bar{P}_6$, que son todas las cadenas que

representan los clanes en H.

Abajo se muestra cómo corta cada enzima de restricción en los sitios de reconocimiento agregados a los oligos que contenían un V_i^1. Las flechas indican donde corta cada enzima.

Enzima Afl II sitio de reconocimiento contenido en $P_0V_0^1P_1$

5'... C$^{\Downarrow}$T-T-A-A-C ... 3'
| | | | | |
3'... G-A-A-T-T$_{\Uparrow}$C ... 5'

Enzima Spe I sitio de reconocimiento contenido en $P_2V_2^1P_3$

5'... A$^{\Downarrow}$C-T-A-G-T ... 3'
| | | | | |
3'... T-G-A-T-C$_{\Uparrow}$A ... 5'

Enzima Hind III sitio de reconocimiento contenido en $\overline{P_2V_1^1P_1}$

5'... A$^{\Downarrow}$A-G-C-T-T ... 3'
| | | | | |
3'... T-T-C-G-A$_{\Uparrow}$A ... 5'

Enzima Sph I sitio de reconocimiento contenido en $\overline{P_4V_3^1P_3}$

5'... G-C-A-T-G$^{\Downarrow}$C ... 3'
| | | | | |
3'... C$_{\Uparrow}$G-T-A-C-G ... 5'

Enzima Stu I sitio de reconocimiento contenido en $P_4V_4^1P_5$

5'... A-G-G$^{\Downarrow}$C-C-T ... 3'
| | | | | |
3'... T-C-C$_{\Uparrow}$G-G-A ... 5'

Enzima Xho I sitio de reconocimiento contenido en $\overline{P_6V_5^1P_5}$

5'... C$^{\Downarrow}$T-C-G-A-G ... 3'
| | | | | |
3'... G-A-G-C-T$_{\Uparrow}$C ... 5'

Los resultados de eliminar cadenas y aplicar PCR son mostrados en las columnas 5 y 6 respectivamente de la Figura 2.4.
Para saber el tamaño de los clanes en la gráfica H basta observar el resultado de electroforesis en gel Figura 2.4. En la columna 6 se observa el tamaño de las cadenas, y por lo tanto se sabrá , por como se construyeron de qué tamaño son los clanes. En este caso la cadena más corta es la importante ya que al tener más 1's es la que tiene menos pares de bases. La más corta es de 160 *bps*, que es la banda más clara, indicada por la flecha en la Figura 2.4. Por lo tanto el tamaño del clan más grande es de 4 vértices, por que son 20 *bps* por los 7 P_i y se suman 20 *bp* de dos ceros que sumaban 10 *bp* por cada uno de éstos.

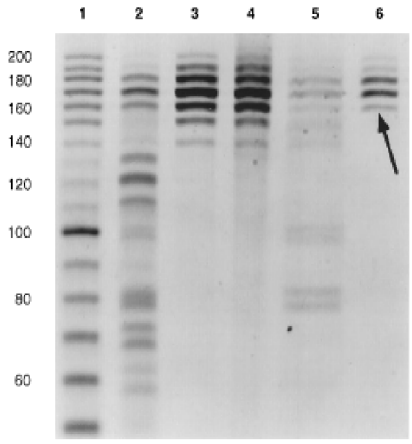

Figura 2.4: Resultado electroforesis en gel

Ya que se sabe el tamaño del clan más grande, además se quisiera saber cuáles son los 4 vértices de los 6 en H, que forman el clan; es decir, cuál de las $15=\binom{6}{4}$ cadenas es la respuesta. Para ésto se utilizó clonación molecular, como se describe a continuación: la cadena de ADN respuesta fue colocada dentro de un virus llamado bacteriófago M13, a través de mutagénesis. En éste proceso se añadió una cola a los extremos del ADN respuesta que es complementaria a las partes del ADN del bacteriófago M13. Se introduce para que la respuesta sea amplificada (PCR con un diseño especial de cebadores). El ADN mutado del bacteriófago M13, el cual contiene la respuesta, fue transferido infectando a la bacteria E. coli y, después de ser clonada, sé extrajó y sé secuencio su ADN.

El resultado fue el esperado, pues se obtuvo la cadena que corresponde a la sucesión `111100`.

6400 ACTCACTATAGGGCGAATTGGGTA CCCTGGATCCCGCCCCTCTC
cola $\overline{P_6}$
GAGTCTATTTCACACGTAGG CCTTTATCTCAAANACTGCA
$\overline{P_5}$ $\overline{P_4}$
TGCGCAGGCTTGTTAGCACT AGTAATCTCTCTTCACGAAG
$\overline{P_3}$ $\overline{P_2}$
CGGTTATGAA CTTACTGATAGGTGACTCTT CTGCCGCGTC
$\overline{V_1^0}$ $\overline{P_1}$ $\overline{V_0^0}$
AAGGTTCGCANAATTCTACG TGGAGCTCCAGCTTTTGTTCC 6260
$\overline{P_0}$ cola

Arriba se muestra el resultado del ADN clonado y secuenciado. El ADN insertado está subrayado y marcado; la letra N indica una base indeterminada, los números indican el lugar de la respuesta en el ADN de bacteriófago M13.

Capítulo 3

Otras aproximaciones para PCM

En este capítulo se verán cinco aproximaciones distintas para el problema del clan máximo. La sección 3.1 presenta un algoritmo genético que es un algoritmo que se implementa en una computadora convencional y el autor trata de dar una interpretación a cada paso para poder implementarlo con ADN. Utilizando estrategias de evolución como la mutación que se aplica con cierta probabilidad, comenzando con una población aleatoria (una muestra de la instancia del problema) con la ventaja de que no necesita crearse la fuente de datos desde un principio. Ésta población se irá mutando conforme avanza el algoritmo y se irán aproximando a un clan máximo. Entonces el autor quiere implementar los pasos del algoritmo de computadora, con cadenas de ADN haciendo que éstas evolucionen al ser mutadas en cada iteración.

Para la sección 3.2 de nuevo la solución se ira creando y no será extraída de una fuente de datos como se vio en el capítulo 2. Todo esto una vez mas con ayuda de enzimas y moléculas de ADN, pero esta vez las moléculas tendrán una forma circular que se llaman plásmidos.

En la sección 3.3 proponen una implementación en un "hardware"para ADN llamado microreactor de micro flujo, donde el flujo serán las cadenas sencillas de ADN. En éste se llevarán acabo las iteraciones para seleccionar clanes válidos en una gráfica, esta selección sera programada ópticamente.

En la sección 3.4 proponen dos algoritmos que al unirlos nos dan el clan máximo de una gráfica, el primero calculando todas las subgráficas inducidas por el número de aristas de un clan de tamaño k y el segundo algoritmo calcula cuáles de estas subgráficas tienen k vértices.

Por último en la sección 3.5 proponen una estructura de membrana, inspirada

en el funcionamiento de las membranas de una célula biológica que sirven como barreras y filtros en el paso de moléculas del interior al exterior de ésta. Entonces implementan la estructura de membrana con un algoritmo basado en ADN, para calcular el clan máximo.

3.1. Cálculo evolutivo para PCM

Como la aproximación anterior está limitada para problemas con a lo más 27 vértices, el autor concluye que es necesario superar esa limitación con un enfoque evolutivo. Entonces Thomas Bäck et al. en 1999 presentan este algoritmo; la solución debe evolucionar en lugar de ser extraída, evitando crear todas las posibles soluciones (fuente de datos) desde el principio para extraer una de éstas. Entonces propone implementar los algoritmos evolutivos usando cadenas de ADN para codificar y evolucionar la solución, por medio de amplificación iterativa y mutación a una población de cadenas. Primero se definirá lo que es cómputo evolutivo enfocado en algoritmos genéticos, y luego el algoritmo evolutivo propuesto para PCM y como implementarlo con cómputo en ADN. Por último se verá el algoritmo evolutivo basado en silicio.

El *cómputo evolutivo* se enfoca en la búsqueda de optimización probabilística y métodos tomados del modelo de evolución orgánica. Los *algoritmos evolutivos*, implementan el proceso de creación de nueva información y su evaluación y selección, donde un individuo de la población es afectado por otros individuos así como por el ambiente. Las mejoras en individuos se realizan bajo tales condiciones, la mayor de éstas es la oportunidad de sobrevivir por un largo tiempo y heredar su información a sus descendientes. Se resume en el siguiente seudocódigo:

Algoritmo evolutivo
t:= 0
`initialize` $(P(t))$ `randomly` $\mid \mathrm{P(t)} \mid = \mu$;
`evaluate` (P(t)) ;
`while not terminate do`
    P'(t) : =`variation` (P(t)), $\mid \mathrm{P(t)'} \mid = \lambda$;
    `evaluate` (P'(t));
    P(t+1) := `select` $_{\mu}$(P'(t) $\cup$ Q);
    t:=t+1;

Donde $P(t)$ denota un multiconjunto en la iteración t de candidatos a la solu-

ción de un problema dado; $P(t)$ es usualmente referido como la población de tamaño μ en la generación t, y los miembros de $P(t)$ son llamados *individuos*. El conjunto Q denota un multiconjunto especial de candidatos de solución que deben ser considerados por selección. Típicamente se elige Q=$\emptyset$, Q=$P(t)$, o Q=$\{\vec{a'}\}$ donde $\vec{a'}$ denota el mejor candidato de solución en $P(t)$. Una población descendiente intermediaria $P'(t)$ de tamaño λ de candidatos de solución es generada por aplicación de operadores tales como recombinación y/o mutación a miembros de la población $P(t)$.

Los *algoritmos genéticos* son una clase particular de algoritmos evolutivos. Comúnmente utilizan una representación binaria de los individuos sobre el alfabeto $\{0, 1\}$. La población inicial $P(0)$, usualmente es creada al azar asignando a cada dígito de los miembros de $P(0)$ uno de los dos valores 0 o 1. Con respecto al operador de variación, se aplica mutación. El operador mutación en algoritmos genéticos es implementado invirtiendo dígitos con una pequeña probabilidad tal como p_m=.001 o $p_m \in [0.005\ ,0.01]$.

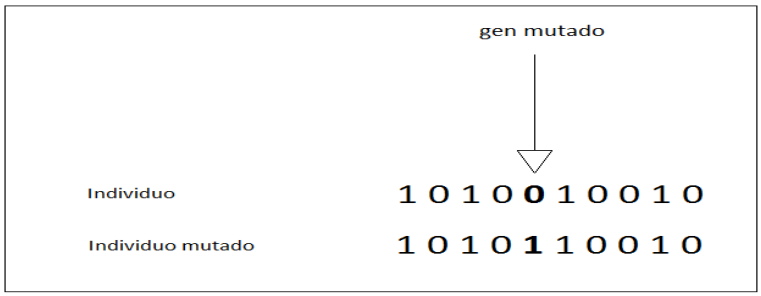

Figura 3.1: Operador de mutación

La implementación del algoritmo utiliza: amplificación con PCR, electroforesis en gel, y mutación que se puede llevar a cabo al explotar los errores de replicación que ocurren en las operaciones biológicas en ADN tal como la amplificación por medio de PCR. También una técnica para realizar mutación por medio de mutagénesis controlada que implica el uso de enzimas como uvrABC y polimerasa.

Algoritmos evolutivos para PCM

Este algoritmo para el problema del clan máximo, puede basarse en amplificación y mutación iterativa a una población de cadenas, removiendo los no-clanes de

la población como en el paso 3 del algoritmo visto en el capítulo 2, y seleccionar las cadenas más cortas por medio de electroforesis en gel, donde una óptima o una solución cercana al óptimo puede ser evolucionada en lugar de extraerla de la población inicial. Utilizará la misma codificación del algoritmo en el capítulo 2. Entonces la mutación del dígito x_i, para pasar de x_i=0 a x_i=1, que en la representación binaria, correspondería a una cancelación a los segmentos correspondientes $10bp$ de V_i de las cadenas ADN; la mutación inversa puede corresponder a introducir V_i. La amplificación de cadenas se llevara a cabo por medio de PCR. Reuniendo todos estos operadores, el algoritmo con cálculo evolutivo basado en ADN está dado por:

Algoritmo evolutivo basado en ADN
`Generate an initial random population` | P(t) | $\ll 2^n$
`while not terminate do`
P:=`amplify and mutate` P;
`Remove the set Y of all non-cliques from` P (i.e., $P := P - Y$);
P:=`select shortest ADN strands from p`;

Nótese que el algoritmo se parece a un (μ , λ)-algoritmo genético con una pequeña μ y un cierto tipo de mutación; es decir, se extraen las μ mejores soluciones de λ candidatos de solución. Se verá en el siguiente algoritmo de computadora, lo más parecido posible a uno basado en ADN, con la idea de obtener y experimentar con un incremento del tamaño en la población. Anteriormente se definió el algoritmo evolutivo, pero a continuación se presenta el algoritmo propuesto por Bäck et al. al problema del clan máximo, que supera la limitación del tamaño del problema con a lo más 27 vértices; dice el autor que "la solución debe evolucionar en lugar de ser extraída combinando cálculo con ADN y algoritmo evolutivo" por medio de amplificación iterativa y mutando una población de cadenas.

Una vez más se eligirá la representación de un candidato de solución como una sucesión binaria (x_1,x_2,...,x_n); $x_i = 1 \Leftrightarrow i \in V'$(conjunto de clanes). Así el i-ésimo dígito indica la presencia (x_i=1) o ausencia (x_i=0) del vértice i en la solución candidato. Donde una sucesión que sea solución candidato, en particular puede representar una solución infactible, pero en este caso es importante que aparezcan para usar una función de penalización que guíe hacia la solución factible. Entonces esta función objetivo ha de clasificar las sucesiones con mayor penalidad más lejos de la factibilidad.

La función de penalización esta dada por:

$$\mathrm{f}(\vec{x}) = \sum_{i=1}^{n} x_i \cdot \left(1 - n \cdot \sum_{j=i+1}^{n} x_j \cdot (1 - e_{ij})\right) \rightarrow max \tag{3.1}$$

Esta función penaliza sucesiones infactibles $\vec{x}$ con una penalización de n para cada arista ausente entre dos nodos i y j en la solución candidato de V' representada por $\vec{x}$. Para sucesiones factibles $\vec{x}$, f($\vec{x}$) ≥ 0 y el valor es dado por el número de nodos en el conjunto independiente representado por $\vec{x}$. Con la ayuda de la matriz de adyacencias $[e_{ij}]$ donde

$$e_{ij} = \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{en otro caso.} \end{cases}$$

Entonces se quiere maximizar la función $f(\vec{x})$, el máximo dará el número de clan máximo, ya que por cada arista ausente el valor de la función se hace más negativo.

Para pasar de una aproximación con cómputo evolutivo, a un cómputo basado en ADN, parece razonable usar el algoritmo más sencillo basado en la población para tratar de explotar el paralelismo masivo del cálculo basado en ADN, llamándolo *(1,λ)-algoritmo,* el cual es un ejemplo del algoritmo siguiente para μ=1.

Algoritmo evolutivo basado en silicio
(μ,λ)-Algoritmo:
$t := 0$;
`initialize` $P(t)$ randomly, $\mid P(t) \mid = \mu$;
`while not terminate do`
    $P(t)' := \emptyset$;
    `for` i:= 1 `to` λ `do`
        $P'(t) := P'(t) \cup \{\texttt{mutate-member}_p(P(t))\}$;
    `evaluate`($P'(t)$);
    $P(t+1) :=$ `select`$_\mu$ ($P'(t)$);
    t:=t+1;

El operador mutar-miembro, con probabilidad p de mutación, aleatoriamente elige un elemento de $P(t)$ y muta estos elementos invirtiendo cada bit con probabilidad p (donde $p \approx \frac{1}{2n} \cdot ln\lambda$), donde una población de λ individuos mutados es generada de $P(t)$ repitiendo este proceso λ veces. Estos individuos mutados son evaluados (calculando sus valores en la función objetivo $f(\vec{x_k})$ dada por la ecuación

3.1 para cada solución $\vec{x_k}$ que representan en $P'(t)$), y el operador selección determinísticamente selecciona los μ mejores individuos de $P'(t)$ para sobrevivir a la siguiente generación (los peores individuos se descartan). Para nuestro ejemplo se simplifica el algoritmo mediante el uso de μ=1 es decir reduciendo el origen de la población a uno. Esto implica que el conjunto intermedio de la población $P'(t)$ consiste de individuos generados por mutación.

Esta aproximación refleja la idea de que el paso de selección, basado en ADN, de computo evolutivo puede ser implementado por medio de electroforesis en gel, seleccionando la más larga o la más corta cadena de ADN.

Justificación del algoritmo

Sea G una gráfica de orden n, el algoritmo comienza eligiendo aleatoriamente una subgráfica inducida por los vértices de K_n. En la primera iteración sólo se evalúa esta subgráfica ya que la probabilidad de modificar ésta es cero, pasando a la siguiente iteración hasta que la probabilidad $p \approx \frac{1}{2n} \cdot ln\lambda$ sea mayor o igual a 0.05. Sí ésto pasa en cada posible vértice en K_n, se agrega o se quita de la subgráfica junto con sus aristas que hace que los una al resto de los vértices en la subgráfica. En el caso de cambiar un 1 por un 0 se quita un vértice en la subgráfica a modificar y por lo tanto las aristas que incidán en él. Y en el caso de cambiar un 0 por 1 se agrega el vértice correspondiente y las aristas que hace que lo unan a todos los vértices en la subgráfica.

En cada iteración se crean n subgráficas de K_n y son evaluadas por la función 3.1 de tal manera que si en su evaluación nos resulta un valor negativo quiere decir que tenía aristas que no pertenecen a G y si el valor es positivo indica el tamaño del clan que ha formando ésta subgráfica. Y como en cada iteración se escoge el más positivo se va acercando cada vez más al óptimo que en este caso sería el clan máximo en G. □

Ejemplo

Se aplicará el algoritmo a la gráfica G de la Figura 3.6. La codificación de las gráficas es igual que en el capítulo anterior. Y se elige $\lambda = 4$ aplicando el operador mutate-member con probabilidad 0.05, el operador select se aplicará por conveniencia a las sucesiones con resultado de la evaluación más positiva y no se hara la selección aleatoria como usualmente se implementa.

Iteración 1 Para $t = 0$ Inicia con $P(0) = \{\texttt{0101}\}$ ésta sucesión es tomada aleatoriamente de la instancia del problema. Entonces $\mid P(0) \mid = 1$ y $P(0)' = \emptyset$

Para $i = 1$ mutar cada bit del elemento en $P(0)$ con probabilidad $p = ln(1)/8$. Se tiene que $P(0)' = P'(0) \cup \{\texttt{mutate-member}_p(P(0))\}$ y por lo tanto $P(0)' = \{\texttt{0101}\}$

Evaluación Se evalúan los elementos de $P'(0)$ con la función dada en 3.1; el resultado de evaluar $\{\texttt{0101}\}$ es -3.

Selección Se selecciona un elemento de $P'(0)$ tal que el resultado de su evaluación sea la más positiva.

Iteración 2 Para $t = 1$ Ahora $P(1) = P'(0)\{\texttt{0101}\}$.

Para $i = 2$ mutar cada bit del elemento en $P(1)$ con probabilidad $p = ln(2)/8$ que es mayor que 0.05. Se tiene que $P(1)' = P'(0) \cup \{\texttt{mutate-member}_p(P(1))\}$ y por lo tanto $P(1)' = \{\texttt{1101, 0001, 0111, 0100, 0101}\}$

Evaluación Se evalúan los elementos de $P'(1)$ con la función dada en 3.1, el resultado de evaluarlos es -5, 1, -1, 1, -3, respectivamente.

Selección Se selecciona un elemento de $P'(1)$ tal que el resultado de su evaluación sea la más positiva (los empates se rompen arbitrariamente).

Iteración 3 Para $t = 2$ Ahora $P(2) = \{\texttt{0100}\}$ y $P'(1) = \{\texttt{1101, 0001, 0111, 0100, 0101}\}$.

Para $i = 3$ mutar cada bit del elemento en $P(2)$ con probabilidad $p = ln(3)/8$ que es mayor que 0.05. Se tiene que $P(2)' = P'(1) \cup \{\texttt{mutate-member}_p(P(2))\}$ y por lo tanto $P(2)' = \{\texttt{0101, 1101, 0001, 0111, 1100, 0110}\}$

Evaluación Se evalúan los elementos de $P'(2)$ con la función dada en 3.1, el resultado de evaluarlos es -3, -5, 1, -1, 2, 2, respectivamente.

Selección Se selecciona un elemento de $P'(2)$ tal que el resultado de su evaluación sea la más positiva (los empates se rompen arbitrariamente).

Iteración 4 Para $t = 3$ Ahora $P(3) = \{\texttt{0110}\}$ y $P'(2) = \{\texttt{0101, 1101, 0001, 0111, 1100, 0110}\}$.

Para $i = 4$ mutar cada bit del elemento en $P(3)$ con probabilidad $p = ln(4)/8$ que es mayor que 0.05. Se tiene que $P(3)' = P'(2) \cup \{\texttt{mutate-member}_p(P(3))\}$ y por lo tanto $P'(3) = \{\texttt{0101, 1101, 0001, 0111, 1100, 0110, 1110, 0010, 0100}\}$

Evaluación Se evalúan los elementos de $P'(3)$ con la función dada en 3.1, el resultado de evaluarlos es -3, -5, 1, -1, 2, 2, 3, 1, 1, respectivamente.

Selección Se selecciona un elemento de $P'(3)$ tal que el resultado de su evaluación sea la más positiva (los empates se rompen arbitrariamente).

Como la gráfica es de 4 vértices y no era completa entonces se puede concluir que tiene un clan máximo de tamaño 3 y es el representado por la sucesión `1110`.

3.2. Escribiendo sobre las moléculas

Este tercer algoritmo, propuesto por Tom Head y Masayuki Yamamura en 1999, para resolver el mismo ejemplo con 6 vértices visto en el capítulo 2, pero empleando un nuevo modelo usando plásmidos, que son moléculas circulares de ADN, para almacenar la información; se llama *moléculas de registro de datos*. Como se observó en el capítulo 1, encontrar el máximo conjunto independiente (el máximo número de vértices que no están conectados entre cualesquiera dos) en una gráfica es equivalente a encontrar el máximo clan. Entonces ahora el objetivo es encontrar el máximo conjunto independiente en la gráfica complementaria $\bar{G}$=$(V, \bar{A})$.

De nuevo el tipo de molécula es elegido de tal manera que contiene lugares específicos distinguibles que representan al 0 y 1; se llama a éstos lugares "*estaciones*". Cada cálculo comienza con cada una de estas moléculas teniendo todas sus estaciones en 1. Conforme el cálculo avanza la configuración de las estaciones son alteradas, en cada iteración cambiaran algunos 1's por 0. Este cambio depende de las aristas que aparecen en la gráfica complementaria: tendrán cero las estaciones que corresponden a vértices que son extremos de las arista en $\bar{G}$. De nuevo la respuesta será el número binario que tenga más unos y este número será el tamaño del máximo conjunto independiente contenido en $\bar{G}$.

En este caso el papel del agua es fundamental. Separa las moléculas permitiendo el acceso a cada una, cambia aleatoriamente el lugar de las moléculas por difusión, permite la partición de las moléculas en un número específico de piezas de forma que se puedan adoptar diferentes acciones en cada parte y permite la reunión de las partes. Esta vez, en las sucesiones binarias el 1 en el lugar i-ésimo quiere decir que pertenece a un conjunto independiente de vértices en una gráfica complementaria y 0 en otro caso; por ejemplo, el conjunto independiente de vértices 2,3,4 en $\bar{H}$ de la gráfica vista en 2.1 es representado por la sucesión `001110`.

Figura 3.2: Molécula de registro de datos

Algoritmo

1 Comenzar con el conjunto inicial $T_{\mathcal{I}}$ que contiene la sucesión de longitud n, donde cada dígito esta representa un 1.

2 Para cada arista (x_i, x_j) en $\bar{A}$, construir dos conjuntos T_i y T_j que contengan el mismo tipo de sucesiones, para la primera arista $T_{\mathcal{I}} = T_i = T_j$ y para las siguientes aristas $T = T_i = T_j$.

3 En paralelo para cada sucesión en T_i convertir a 0 en el lugar del dígito que corresponde al vértice x_i y convertir un 0 en cada sucesión de T_j en el lugar del dígito que corresponde al vértice x_j.

4 Unir los elementos de T_i y T_j en un sólo conjunto T, regresar al paso 2.

5 Cuando ya se hayan puesto todos los 0's por cada arista en $\bar{G}$, contar los 1's en cada sucesión contenida en T.

Justificación del algoritmo

Esta vez el objetivo del algoritmo es encontrar el máximo conjunto independiente en $\bar{G}$. Sea $\bar{G}$ la gráfica complementaria de G, de orden n y tamaño m. Para el primer paso del algoritmo se tiene la gráfica $\bar{K}_n$, en el paso dos y tres en cada iteración se van eliminando vértices, para construir subconjuntos de vértices independientes en $\bar{G}$. Se eliminan los vértices tal que $(x_i, x_j) \in A(\bar{G})$, en cada iteración quedan dos tipos de subconjunto de vértices; los que no contiene a x_i y los que no contienen a x_j, ya que no pueden pertenecer a un subconjunto independiente si los une una arista por definición. Después de todas las iteraciones, una por cada arista de los pasos del 2 al 4, lo que resulta son todos los posibles subconjuntos de vértices independientes en $\bar{G}$. Supóngase que alguno de estos subconjuntos de

vértices no es un subconjunto independiente, entonces existen al menos dos vértices adyacentes en $\bar{G}$, pero por los pasos 2 al 4, que se iteraron por cada arista en $\bar{G}$, se eliminaron los subconjuntos de vértices que contienen al mismo tiempo estos vértices que se supusieron adyacentes.**!**. Por lo tanto todos los subconjuntos de vértices en K_n que quedaron son subconjuntos de vértices independientes en $\bar{G}$. Por último se cuentan los vértices de cada subconjunto y se toma el máximo □.

Ejemplo:

Tomando la misma gráfica H de la Figura 2.1 con $n = 6$, vista en el capítulo 2.

Paso 1 Sea $T_{\mathcal{I}} = \{\texttt{111111}\}$.

Paso 2 Para la arista (0,2) en $\bar{A}$ crear $T_{\mathcal{I}} = T_i = T_j = \{\texttt{111111}\}$.

Paso 3 En la sucesión que contiene T_i convertir a cero en el lugar del vértice 0 quedando de la forma $T_i = \{\texttt{011111}\}$ y en conjunto T_j se convierte un cero en el lugar del vértice 2 resultando $T_j = \{\texttt{110111}\}$

Paso 4 Unir los conjuntos T_i y T_j en $T = \{\texttt{011111,110111}\}$ y regresar al paso 2.

Paso 2 Para la arista (0,5) en $\bar{A}$ crear $T = T_i = T_j = \{\texttt{011111,110111}\}$.

Paso 3 Por cada sucesión en T_i convertir a cero en el lugar del vértice 0, en caso de ya estar en cero se deja igual, entonces $T_i = \{\texttt{011111,010111}\}$. En el conjunto T_j convertir a cero en el lugar del vértice 5, resultando $T_j = \{\texttt{011110,110110}\}$

Paso 4 Unir los conjuntos T_i y T_j en $T = \{\texttt{011111,010111,011110,110110}\}$ y regresar al paso 2.

Paso2 Para la arista (1,5) en $\bar{A}$ crear $T = T_i = T_j = \{\texttt{011111,010111,011110,}$ $\texttt{110110}\}$.

Paso 3 Por cada sucesión en T_i convertir a cero en el lugar del vértice 1, entonces $T_i = \{\texttt{001111,000111,001110,100110}\}$. En el conjunto T_j convertir a cero en el lugar del vértice 5, resultando $T_j = \{\texttt{011110,010110,110110}\}$

Paso 4 Unir los conjuntos T_i y T_j en $T = \{\texttt{001111,000111,001110,100110,}$ $\texttt{011110,010110,110110}\}$ y regresar al paso 2.

Paso 2 Para la arista (1,3) en $\bar{A}$ crear $T = T_i = T_j = \{\texttt{001111,000111,001110,}$ $\texttt{100110,011110,010110,110110}\}$.

Paso 3 Por cada sucesión en T_i convertir a cero en el lugar del vértice 1, entonces $T_i = \{$`001111,000111,001110,100110,000110`$\}$. En el conjunto T_j convertir a cero en el lugar del vértice 3, resultando $T_j = \{$`001011,000011, 001010,100010,011010,010010,110010`$\}$

Paso 4 Unir los conjuntos T_i y T_j en $T = \{$`001111,000111,001110,100110, 000110,001011,000011,001010,100010,011010,010010,110010`$\}$ como ya no hay más aristas en $\bar{A}$ seguir con el paso 5.

Paso 5 Contar los 1's en cada sucesión contenida en T que son 4 en la sucesión `001111` entonces el máximo conjunto independiente en $\bar{H}$ es de tamaño 4 y lo forman los vértices 2,3,4,5, de igual manera estos vértices forman el clan máximo en H.

Implementación

En el paso uno del algoritmo, comienza con muchas moléculas idénticas llamadas plásmidos que son cadenas dobles de ADN en forma circular (moléculas de registro de datos), contenidas en un tubo T_I. Estas contienen subsegmentos, MCS (*sitios de clonación múltiple*) de aproximadamente $175bp$, que a su vez contienen n sitios ajenos (estaciones, Figura 3.2), cada uno representando al dígito 1, que podrán ser modificados usando de nuevo enzimas de restricción cambiando a la estación para representar al dígito 0.

En el segundo paso, por ejemplo para la primera arista (x_i, x_j) en $\bar{A}$, se divide en cantidades iguales en dos tubos T_i y T_j las moléculas contenidas en T_I. En el tercer paso, para T_i se añade la enzima que modifica a todas las moléculas contenidas, haciendo que ahora se represente al 0 en la estación correspondiente al vértice x_i, y en T_j añadir otra enzima distinta que pone un 0 en el lugar del vértice x_j.

En el cuarto paso se une el contenido de los tubos T_i y T_j en un tubo T que contendrá dos tipos de moléculas las que representan 0 en la estación i-ésima y en las otras estaciones el 1 y las que representan 0 en el lugar j-ésimo y 1 en las otras estaciones. El tubo T que ya contiene los dos tipos de moléculas se regresa al paso 2 dividiendo su contenido en cantidades iguales en los tubos T_i y T_j para luego modificar estas nuevas moléculas de registro con la información de la siguiente arista en $\bar{A}$. Así regresar a los pasos del 2 hasta el 4 por cada arista en $\bar{A}$. Terminando ésto viene el paso 5 que es contar el número de 1's en cada molécula distinta contenida en T.

Se convierte a cero una estación, alterando el sitio realizando los siguientes tres pasos: 1) Linealizar el plásmido, es decir se hace un corte en la estación correspondiente a convertir a cero, esto lo realiza la enzima de restricción asociada al sitio.

2) Usando ADN polimerasa se extiende el final $3'$ de la cadena, produciendo una molécula lineal. 3) Aplicando una ligasa para volver a formar un círculo con la molécula lineal ligando los finales.

Cuando una estación es alterada para representar el 0, la longitud de la estación se incrementa por cuatro pares de bases, una vez que son alterados, en los cálculos posteriores ya no serán alterados.

Por último para calcular el máximo número de unos, se realizan los siguientes tres pasos: 1) Utilizando un par de enzimas de restricción, se cortan los plásmidos en su sitio de clonación múltiple, cada una corta uno de los extremos de MCS. 2) Separar las cadenas cortas de longitud $175bp$ aproximadamente con electroforesis en gel. 3) De las moléculas que tienen la mínima longitud en pares de bases, calcular el número de sitios de enzimas de restricción que quedaron como a continuación se muestra.

Por ejemplo si la longitud en pares de bases de las moléculas de longitud mínima es 187 entonces el máximo número de unos es $6 - (1/4)(187 - 175) = 3$. Por que la operación "convertir a 0" hace que el plásmido aumente en 4 pares de bases en el lugar de la estación a modificar, entonces si la mínima es de $187bp$ se le resta $175bp$ que ya tenía el MCS, lo que resulta se divide entre cuatro para ver cuántas estaciones se convirtieron a ceros. Suponiendo que las estaciones eran 6 que son el número de 1's al comienzo, se resta el número de ceros y da como resultado el número de 1's en la molécula de longitud mínima.

3.3. Cómputo con ADN en microreactores

En el 2000 John S. McCaskill, propone un algoritmo sencillo para seleccionar subconjuntos correspondientes a clanes, enfocándose en el problema de cómo programar la selección de clanes válidos y dejando a un lado el problema de síntesis de cadenas. Esta vez la implementación es en un *microreactor*; éste es, un dispositivo en donde las reacciones químicas tienen lugar en microcanales con medidas menores a 1 mm, los cuales pueden ser programados ópticamente. En este caso las reaciones químicas son naturalización y desnaturalización. No se tendrá la necesidad de usar enzimas de restricción ni manipulaciones con tubos en el laboratorio. Se implementará una técnica para seleccionar una cadena sencilla de ADN con una secuencia específica, usando *balines magnéticos*, creando los oligos complementarios, que serán sujetados por los balines diminutos, cuando los oligos complementarios se unan a las cadenas objetivo. Entonces estos microreactores tendrán unos *Módulos*

de Selección de Transferencia (STM por sus siglas en inglés), con un interruptor magnético para seleccionar cadenas con una secuencia específica e implementa la operación básica de manipular un gran número de moléculas de ADN en paralelo bajo un flujo constante.

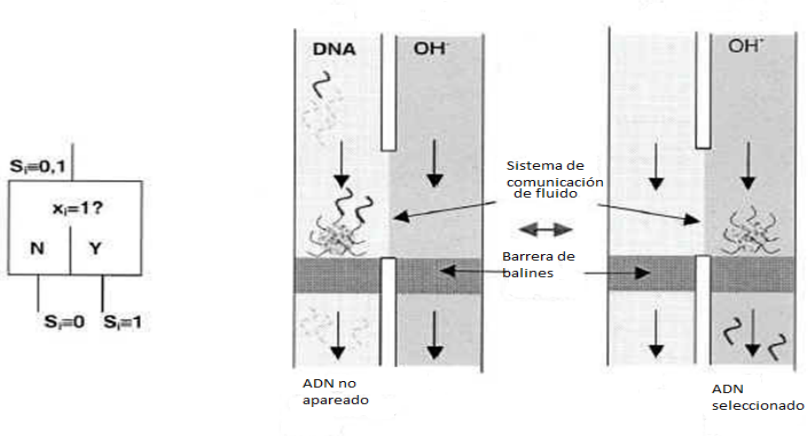

Figura 3.3: Módulo de selección de transferencia STM.

En la Figura 3.3 se tiene un módulo de selección de transferencia. El lado izquierdo muestra el uso de símbolos para el diseño del módulo que selecciona cadenas de ADN teniendo una secuencia S_i en la posición i correspondiente al valor x_i=1. El lado derecho muestra el flujo de fluido del módulo de selección. El STM es el que lleva a cabo la selección de clanes, por medio de balines magnéticos. Cadenas de ADN no apareadas con los balines fluyen directo a través del canal izquierdo del STM. Y las cadenas que hibridaron a la secuencia asociada al balín, son removidas al canal del lado derecho por medio de un imán que atrae al balín, para luego ser liberadas las cadenas seleccionadas. Este sistema resuelve ejemplos arbitrarios de orden $n \leq 100$. La gráfica en específico es codificada en un patrón inicial; cada módulo es asociado a uno de los $2n$ oligonucleótidos.

Algoritmo

El algoritmo para encontrar el máximo clan puede ser dividido en dos partes: Sea $G = (V, A)$ una gráfica, con vértices $V = \{x_1, x_2, ..., x_n\}$.

1 Encontrar todos los subconjuntos de vértices en la gráfica K_n.

- Para cada vértice x_i ($i \geq 1$) en la gráfica retener sólo los subconjuntos o que no contengan al vértice i o teniendo sólo otros vértices x_j tales que la arista (i,j) está en la grafica.

2 Encontrar uno de los más grandes.

Justificación del algoritmo

Este algoritmo funciona por que en el paso i-ésimo al retener a todos los subconjuntos que no contengan al vértice x_i, se retienen a todos los posibles clanes en G con $n-1$ vértices, de los cuales puede haber subconjuntos que no representen clanes que pertenezcan a G. Pero éstos se van descartando en los pasos siguientes, ya que al retener a los subconjuntos que contengan los vértices x_j, tal que $(x_i, x_j) \in A$ se esta reteniendo los subconjuntos que contienen al vértice x_i, pero no a cualquiera que lo contenga, sólo aquellos que forman clanes contenidos en G, con los vértices x_i y los x_j. Si alguna arista de la forma $(x_j, x_{j'})$ no pertenece a G, con $(x_i, x_{j'}) \in A$ los subconjuntos que sean retenidos con estos tres vértices en el paso j o j' son removidos, así se eliminan todos los subconjuntos que no son clanes de G. Al final quedaran todos los clanes contenidos en la gráfica .

Ejemplo:

Se seguirá con el ejemplo de la gráfica H Figura 2.1, vista en el capítulo 2, usando la misma codificación para representar clanes. Se comienza creando todas las sucesiones binarias que representan todos los posibles clanes en K_6 (ver paso 1 en el ejemplo del capítulo 2).

Paso 1 Seleccionar todos los clanes contenidos en H.

Iteración 1 Retener todas las sucesiones que no contengan al vértice 0, es decir, las sucesiones que tienen cero en el primer dígito. Se tiene el siguiente conjunto:

`000000,000001,000010,000100,001000,010000,001010,011000,001100,`
`000110,000011,010001,001110,001001,000101,010010,010100,001101,`
`010011,001011,010101,001111,011011,011101,011110,011111,010111,`
`011010,011100,010110,011001,000111.`

También retener todas las sucesiones que teniendo al vertice 0, también tienen los vértices 1 , 3, ó 4, que son los que forman aristas con el vértice 0 en H. En total se retuvieron todas las sucesiones que tienen

un 1 en el lugar del dígito 0, 1, 3, 4, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

100000,110000,110100,110110,010000,010100,010110,000100,000110,
000010,100100,100010,010010,110010,100110.

Iteración 2 De las sucesiones retenidas anteriormente seguir con la selección, retener a todas aquellas que no contengan al vértice 1, es decir que tengan cero en el segundo dígito. Obteniendo el siguiente conjunto:

000000,000001,000010,000100,001000,100000,001010,001100,000110,
000011,001110,001001,000101,001101,001011,001111,000111,100100,
100010.

Retener a todas las sucesiones que contengan a los vértices 0, 2, 4, que son los que forman aristas con el vértice 1 en H y también pueden contener al vértice 1. Entonces se retienen todas las sucesiones que tienen un 1 en el lugar del dígito 0, 1, 2, 4, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

100000,010000,001000,000010,110000,011000,011010,110010,100010,
001010,010010,100010.

Iteración 3 De las sucesiones retenidas anteriormente seguir con la selección, retener a todas aquellas que no contengan al vértice 2, es decir que tengan cero en el tercer dígito. Obteniendo el siguiente conjunto:

000000,000001,000010,000100,100000,110000,000110,000011,100100,
100010,010010,110010,000111.

Retener a todas las sucesiones que contengan a los vértices 1, 3, 4, 5 que son los que forman aristas con el vértice 2 en H y también pueden contener al vértice 2. Entonces se retienen todas las sucesiones que tienen un 1 en el lugar del dígito 1, 2, 3, 4, 5, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

001000,001010,001110,001100,001001,001101,001011,001111,011010,
000001,000010,000100,100000,000011,000110,110000,000101,100100,

100010,010010,110010,000111.

Iteración 4 De las sucesiones retenidas anteriormente seguir con la selección, retener a todas aquellas que no contengan al vértice 3, es decir que tengan cero en el cuarto dígito. Obteniendo el siguiente conjunto:

000000,000001,000010,001000,100000,001010,110000,000011,110010,
010010,100010,001001,001011,011010.

Retener a todas las sucesiones que contengan a los vértices 0, 2, 4, 5 que son los que forman aristas con el vértice 3 en H y también pueden contener al vértice 3. Entonces se retienen a todas las sucesiones que tienen un 1 en el lugar del dígito 0, 2, 3, 4, 5, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

000001,000010,001000,001010,000011,010010,001001,001011,011010,
000111,001111,001101,000101,001110,000110,001100,000100.

Iteración 5 De las sucesiones retenidas anteriormente seguir con la selección, retener a todas aquellas que no contengan al vértice 4, es decir que tengan cero en el quinto dígito. Obteniendo el siguiente conjunto:

000000,000001,000100,001000,100000,110000,001100,001001,000101,
001101.

Retener a todas las sucesiones que contengan a los vértices 0, 1, 2, 3, 5 que son los que forman aristas con el vértice 4 en H y también pueden contener al vértice 4. Entonces se retienen todas las sucesiones que tienen un 1 en el lugar del dígito 0, 1, 2, 3, 4, 5, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

000001,000100,001000,100000,110000,001100,001001,000101,000010,
001010,000110,000011,001110,100010,010010,110010,001011,001111,
011010,000111.

Iteración 6 De las sucesiones retenidas anteriormente seguir con la selección, retener a todas aquellas que no contengan al vértice 5, es decir que tengan cero en el sexto dígito. Obteniendo el siguiente conjunto:

`000000,000010,000100,001000,100000,001010,110000,001100,000110,`
`001110,100010,010010,110010,011010.`

Retener a todas las sucesiones que contengan a los vértices 2, 3, 4 que son los que forman aristas con el vértice 5 en H y también pueden contener al vértice 5. Entonces se retienen todas las sucesiones que tienen un 1 en el lugar del dígito 2, 3, 4, 5, estas sucesiones pueden intersectar al conjunto anterior quedando el siguiente conjunto:

`000001,000010,000100,001000,001010,001100,000110,000011,001110,`
`001001,000101,001101,001011,001111,000111.`

Paso 2 Encontrar de las sucesiones anteriores la que contenga más 1's, que es la sucesión `001111` representando al clan formado por los vértices 2, 3, 4, 5.

Implementación

Dada una gráfica G con vértices numerados de 1 a n, se tomará la misma codificación de los vértices adoptada en Ouyang et al. En la construcción de cadenas que representan todos los clanes en K_n se usara POA. Pero esta vez, sólo se requieren de $O(n)$ pasos para construir las 2^n cadenas que representan los subconjuntos de vértices, en contraste con Ouyang et al. que usa $O(n^2)$ pasos. Éstos se introducen en un microreactor, el cual contiene $3n(n-1)/2$ módulos de selección de trasferencia (STM) (ejemplo Figura 3.4), donde se retienen todas las cadenas de ADN con una secuencia específica, por medio de balines magnéticos como se vio en la Figura 3.3.

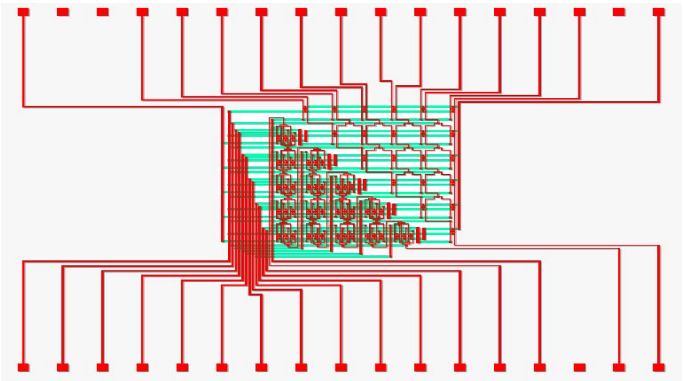

Figura 3.4: Diseño de microreactor para una gráfica con 6 vértices

Cada paso de selección consiste de tres módulos de selección conectados en paralelo. Para el primer paso del algoritmo, esto puede ser implementado en dos bucles anidados (sobre i y j), cada paso involucrando dos selectores en paralelo. Y un tercer selector se introduce para permitir la selección de secuencias fijadas independientemente de la gráfica. Figura 3.5

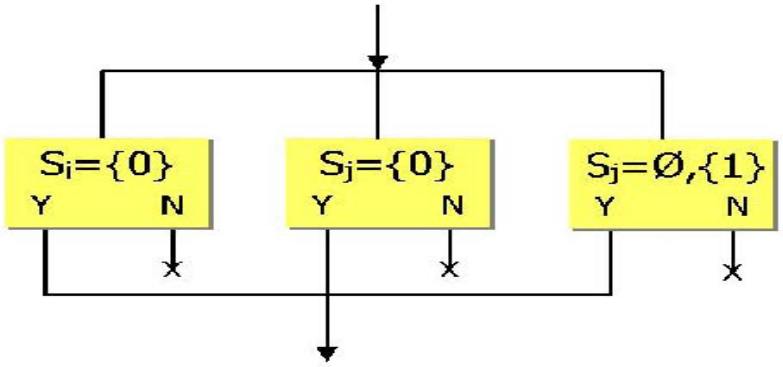

Figura 3.5: Diagrama de flujo que muestra el paso de selección para los subconjuntos de vértices. El primer módulo muestra la ausencia de i el segundo módulo la ausencia de$j$ y el tercero la presencia de la arista (i, j)

Los módulos de selección de transferencia STM pueden ser programados ópticamente, es decir el microreactor es iluminado con luz UV a través de una foto máscara que es una placa opaca con agujeros o transparencias que permiten que la luz brille a través de un patrón definido. En nuestro caso este patrón es la matriz de adyacencias de la gráfica. Se abre por encima del correspondiente STM. El modelo de la máscara es una conversión directa de la matriz binaria representando las aristas de la gráfica G (la entrada a_{ij} es 1 si la arista (i, j) está en G y 0 en otro caso, además es 1 en a_{ij} si $i = j$). Entonces la luz UV láser determina si son o no retenidas las cadenas en el balín asociado a cada STM. Si un par de vértices no está conectado en G, el correspondiente tercer módulo STM no retiene las cadenas. Sólo se programa el tercer módulo en cada paso de selección. Después de cada paso de selección la sub-población es pasada dentro del siguiente paso de selección. De esta manera, siguiendo con el algoritmo de selección, al final la población de cadenas de ADN consistirá de todos los posibles clanes representados en la gráfica dada.

Para determinar el máximo clan, se puede utilizar electroforesis en gel o un procedimiento de clasificación, para seleccionar la cadena de ADN con el mayor

número de 1's representados en sus secuencias. Esta selección se realizará por medio de otros módulos de clasificación contenidos en el microreactor. Estos módulos clasifican las cadenas por la longitud del clan. Si se pudiera ver la trayectoria de las cadenas etiquetándolas fluorescentes (con una cierta clase de molécula fluorescente conocida como fluorophore) a través del módulo de selección, no se necesitaría electroforesis en gel para analizar la cadena.

3.4. Un etiquetado eficiente de ADN

En el 2002, Karl-Heinz Zimmermann presenta algoritmos eficientes con etiquetado de ADN para problemas de gráficas NP-completos; entre ellos, el problema del clan máximo. Se dice que son eficientes por que se realizan en un número de pasos total a $4m + 2n(k+1) - k^2 - k$ en el caso de PCM, donde n es el número de vértices, m es el número de aristas y k es el número de clan. Primero se definirá en qué consiste el etiquetado de ADN, para luego continuar con dos subrutinas que serán parte del algoritmo para el problema del clan máximo.La primera arroja todas las subgráficas inducidas con $\binom{k}{2}$ número de aristas (ver el ejemplo en la Figura 3.6). La segunda subrutina, de todas las subgráficas inducidas por aristas escoge las que tienen k vértices.

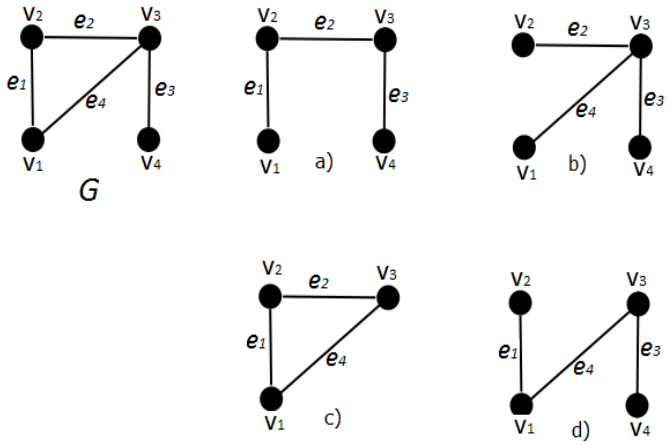

Figura 3.6: G una gráfica y a), b), c), d) son todas sus subgráficas inducidas por $\binom{3}{2}$=3 aristas. Entonces la única subgráfica con tres vértices y 3 aristas es c) el clan máximo en este caso.

Modelo de etiquetado

El *etiquetado de ADN* es un modelo de cómputo molecular, el cual hace uso de cadenas de ADN para representar la información. Esta vez no se requiere de la extensión de cadenas. La *memoria* de un modelo de etiqueta consiste de una cadena de ADN que es parcialmente doble y representa números binarios de longitud n. Cada memoria está formada por dos tipos de cadena sencilla de ADN, moléculas referidas como cadenas de memoria y cadenas de etiqueta. Concretamente una *cadena de memoria* es una ssADN molécula de l bases de longitud, que contiene n subcadenas sin traslapo, cada una de las cuales tiene m bases de longitud llamadas *cadenas de etiqueta*, por lo tanto $l = mn$. Se requiere que cada cadena de etiqueta sea complementaria exactamente a una de las n subcadenas en una cadena de memoria. Cada subcadena de una cadena de memoria representará un dígito en un número binario. Si una cadena de etiqueta reconoce a su subcadena complementaria en la cadena de memoria, se define como "1", en otro caso es como "0". En la siguiente cadena de memoria, cada barra separa una cadena de etiqueta de longitud $m = 6$ (y por tanto n=4).

5' AAAAAA | TTTTTT | GGGAAA | CCCTTT 3'

Abajo se muestran tres ejemplos de memoria compleja que representan a los números binarios `0000,1010, 0111,` respectivamente.

1) AAAAAA | TTTTTT | GGGAAA | CCCTTT

2) AAAAAA | TTTTTT | GGGAAA | CCCTTT
   TTTTTT CCCTTT

3) AAAAAA | TTTTTT | GGGAAA | CCCTTT
   AAAAAA | CCCTTT | GGGAAA

Ya que se tiene la memoria compleja, el cómputo del modelo de etiquetado consiste de una secuencia finita de operaciones como *mezclar, separar, poner, y limpiar*. Las subrutinas, aceptarán como entrada una biblioteca que denotaremos como $[m,\binom{n}{k}]$, donde $1 \leq k \leq n$. Cada biblioteca $[m,\binom{n}{k}]$ consiste de memoria compleja que contiene m subcadenas las cuales corresponden a las palabras binarias de la forma $w0^{m-n}$, donde w es una palabra binaria de longitud n con k subcadenas de 1s y $n-k$ subcadenas de 0s y 0^{m-n} es una palabra de ceros de longitud $m-n$. Así se produce una codificación de todos los subconjuntos de tamaño k de un conjunto con n elementos.

Sea $G = (V, E)$ una gráfica con $V = \{x_1, ..., x_n\}$ el conjunto de vértices y $E = \{e_1, ..., e_m\}$ el conjunto de aristas. La i-ésima arista es denotada por $e_i = \{x_{i1}, x_{i2}\}$. El primer algoritmo de etiqueta que se revisará produce todas las subgráficas inducidas por un k-subconjunto de aristas de G, donde $1 \leq k \leq m$. La entrada del algoritmo es una $[m + n, \binom{n}{k}]$ biblioteca. N_0 proporciona la codificación de todos los k-subconjuntos de aristas.

```
Edge Induced Graphs (N0, m, n)
      for i ← 1 a m do
            separate +(N0, i) y −(N0, i)
            N+ ← set (+(N0, i), m + i1)
            N+ ← set (N+, m + i2)
            N0 ← merge (N+, −(N0, i) )
      return N0
   end Edge Induced Graphs
```

Las operaciones utilizadas en este algoritmo son las siguientes:

1. *Separate,* para un tubo N y un entero i, $1 \leq i \leq n$, se producen dos nuevos tubos, el tubo $+(N, i)$ y el tubo $-(N, i)$. El tubo $+(N, i)$ consiste de aquellas memorias complejas del tubo N en la cual la i-ésima subcadena es 1, y el tubo $-(N, i)$ consiste de aquellas memorias complejas de el tubo N las cuales si i-ésima subcadena es 0. En su implementación esta operación requiere hibridación, se agregan por la pared del tubo, cadenas sencillas diseñadas de tal manera que sean complementarias a la i- esima subcadena.

2. *Set*, para un tubo N y un entero i, $1 \leq i \leq n$, genera un nuevo tubo (N, i) en el cual la i-ésima subcadena de cada memoria compleja del tubo N se convertiran los 0 en 1 y los 1 son dejados igual. También es implementado por hibridación, una gran cantidad de cadenas de etiqueta asociado a la i-ésima posición es agregado al tubo.

3. *Merge*, hace que la memoria compleja de dos tubos de entrada, sean combinados dentro de un tubo. Esta operación es realizada vertiendo el contenido de los dos tubos dentro de un tercer tubo.

Ejemplo:

Aplicando el algoritmo anterior a la gráfica de la Figura 3.6, como $m = 4$, $n = 4$ y $k = 3$, entonces nuestra biblioteca es de $[8, \binom{4}{3}]$ y la forman las palabras $w0^4$, donde w es una palabra binaria de longitud 4 con tres 1s y un 0, 0^4 es una palabra de ceros de longitud 4. Los primeros 4 dígitos corresponden a las 4 aristas,

e_1, e_2, e_3, e_4 y los siguientes cuatro dígitos corresponden a los 4 vértices v_1, v_2, v_3 y v_4, en la gráfica G. Por ejemplo la sucesión 11101111 corresponde a la subgráfica $G \setminus e_4$ representada en a).

Biblioteca={11100000,10110000,11010000,01110000}

Iteración 1 Para $i = 1$

- Separate: en $+(N_0, 1)$ poner las sucesiones que tienen un 1 en el primer dígito y en $-(N_0, 1)$ poner las sucesiones que tienen 0 en el primer dígito. $+(N_0, 1)$={11100000,10110000,11010000} y $-(N_0, 1)$={01110000}
- Set: en el lugar del dígito $4{+}1_1$ poner un 1 a los elementos del conjunto $+(N_0, 1)$, aqui 1_1 representa el primer vértice de la primera arista; nombrado, el vertice 1. N^+={11101000,10111000,11011000}
- Set: en el lugar del dígito $4{+}1_2$ poner un 1 a los elementos del conjunto N^+ aqui 1_2 representa el primer vértice de la primera arista; nombrado, el vertice 2.
  N^+={11101100,10111100,11011100}
- Merge: Es unir los elementos de N^+ con los elementos de $-(N_0, 1)$
  N_0={11101100,10111100,11011100,01110000}

Iteración 2 Para $i = 2$

- Separate: en $+(N_0, 2)$ poner las sucesiones que tienen un 1 en el segundo dígito y en $-(N_0, 2)$ poner las sucesiones que tienen 0 en el segundo dígito. $+(N_0, 2)$={11101100,11011100,01110000} y $-(N_0, 2)$={10111100}
- Set: en el lugar del dígito $4{+}2_1$ poner un 1 a los elementos del conjunto $+(N_0, 2)$
  N^+={11101100,11011100,01110100}
- Set: en el lugar del dígito $4{+}2_2$ poner un 1 a los elementos del conjunto N^+
  N^+={11101110,11011110,01110110}
- Merge: Es unir los elementos de N^+ con los elementos de $-(N_0, 2)$
  N_0={11101110,11011110,01110110,10111100}

Iteración 3 Para $i = 3$

- Separate: en $+(N_0, 3)$ poner las sucesiones que tienen un 1 en el tercer dígito y en $-(N_0, 3)$ poner las sucesiones que tienen 0 en el tercer dígito. $+(N_0, 3)$={11101110,01110110,10111100} y $-(N_0, 3)$={11011110}

- Set: en el lugar del dígito $4{+}3_1$ poner un 1 a los elementos del conjunto $+(N_0, 3)$
  N^+={11101110,01110110,10111110}
- Set: en el lugar del dígito $4{+}3_2$ poner un 1 a los elementos del conjunto N^+
  N^+={11101111,01110111,10111111}
- Merge : Es unir los elementos de N^+ con los elementos de $-(N_0, 3)$
  N_0={11101111,01110111,10111111,11011110}

Iteración 4 Para $i = 4$

- Set: en $+(N_0, 4)$ poner las sucesiones que tienen un 1 en el cuarto dígito y en $-(N_0, 4)$ poner las sucesiones que tienen 0 en el cuarto dígito $+(N_0, 4)$={01110111,10111111,11011110} y $-(N_0, 4)$={11101111}
- Set: en el lugar del dígito $4{+}4_1$ poner un 1 a los elementos del conjunto $+(N_0, 3)$
  N^+={01111111,10111111,11011110}
- Set: en el lugar del dígito $4{+}4_2$ poner un 1 a los elementos del conjunto N^+
  N^+={01111111,10111111,11011110}
- Merge: Es unir los elementos de N^+ con los elementos de $-(N_0, 4)$
  N_0={01111111,10111111,11011110,11101111}

Paso final Lo que quedó en el último conjunto
N_0={01111111,10111111,11011110,11101111} corresponde a las subgráficas inducidas por tres aristas de G en la Figura 3.6 b), d), c), a), respectivamente.

A continuación se presenta la segunda subrutina introducida al algoritmo para encontrar el clan máximo, que consiste en extraer de un tubo N_0, que contienen aquellas memorias complejas en las cuales exactamente k de las subcadenas $m+1$, ... $m+n$ son 1's.

```
Weightening (N_0, m, n, K)
     for i ← 0 a n − 1 do
       for j ← i down to 0 do
             separate +(N_j, m + i + 1) and −(N_j, m + i + 1)
             N_{j+1} ← merge (+(N_j, i + 1), N_{j+1})
             N_j ← −(N_j, i + 1)
     return N_k
end Weightening
```

Ejemplo:

La entrada de este algoritmo será el conjunto N_0, resultado del ejemplo anterior. Este nuevo algoritmo proporciona cuantas de las sucesiones representan subgráficas de G con tres aristas y tres vértices; es decir, las que representan a K_3.

Iteración 1

Para i=0

j=0 Separate $+(N_0, 5)$ y $-(N_0, 5)$
$+(N_0, 5)$=$\{$`01111111,10111111,11011110,11101111`$\}$, este conjunto son todas las sucesiones en N_0 con 1 en la quinta posición y $-(N_0, 5)$=$\emptyset$ es el conjunto de sucesiones en N_0 con 0 en la quinta posición.
N_1:$+(N_0, 5) \cup N_1$=$\{$`01111111,10111111,11011110,11101111`$\}$ este conjunto es la unión de las sucesiones en $+(N_0, 5)$ y N_1
N_0:$-(N_0, 5)$=$\emptyset$

Iteración 2

Para i=1

j=1 Separate $+(N_1, 6)$ y $-(N_1, 6)$
$+(N_1, 6)$=$\{$`01111111,10111111,11011110,11101111`$\}$, este conjunto son todas las sucesiones en N_1 de la iteración 1, con 1 en la sexta posición y $-(N_0, 5)$=$\emptyset$ es el conjunto de sucesiones en N_1 de la iteración 1, con 0 en la sexta posición.
N_2:$+(N_1, 6) \cup N_2$=$\{$`01111111,10111111,11011110,11101111`$\}$ este conjunto es la unión de las sucesiones en $+(N_1, 6)$ y N_2
N_1:$-(N_1, 6)$=$\emptyset$

j=0 Separate $+(N_0, 6)$ y $-(N_0, 6)$
$+(N_0, 6)$=$\emptyset$, este conjunto son todas las sucesiones en N_0 el conjunto de la iteración anterior para $j = 1$, con 1 en la sexta posición y $-(N_0, 6)$=$\emptyset$ es el conjunto de sucesiones en N_0 el conjunto de la iteración anterior para $j = 1$, con 0 en la sexta posición.
N_1:$+(N_0, 6) \cup N_1$=$\emptyset$ este conjunto es la unión de las sucesiones en $+(N_0, 6)$, y N_1 el conjunto de la iteración anterior para $j = 1$.
N_0:$-(N_0, 6)$=$\emptyset$

Iteración 3

Para $i = 2$

j=2 Separate $+(N_2, 7)$ y $-(N_2, 7)$
$+(N_2, 7)$=`{01111111,10111111,11011110,11101111}`, este conjunto son todas las sucesiones en N_2 de la iteración 2, con 1 en la séptima posición y $-(N_2, 7) = \emptyset$ es el conjunto de sucesiones en N_2 de la iteración 2, con 0 en la séptima posición.
$N_3 : +(N_2, 7) \cup N_3$=`{01111111,10111111,11011110,11101111}` este conjunto es la unión de las sucesiones en $+(N_2, 7)$ y N_3
N_2:$-(N_2, 7)=\emptyset$

j=1 Separate $+(N_1, 7)$ y $-(N_1, 7)$
$+(N_1, 7)=\emptyset$, este conjunto son todas las sucesiones en N_1 de la iteración 2, con 1 en la séptima posición y $-(N_1, 7)=\emptyset$ es el conjunto de sucesiones en N_1 de la iteración 2, con 0 en la séptima posición.
N_2:$+(N_1, 7) \cup N_2=\emptyset$ este conjunto es la unión de las sucesiones en $+(N_1, 7)$, y N_2 de la iteración anterior para $j = 2$
$N_1 : -(N_1, 7) = \emptyset$

j=0 Separate $+(N_0, 7)$ y $-(N_0, 7)$
$+(N_0, 7)=\emptyset$, este conjunto son todas las sucesiones en N_0 de la iteración 2, con 1 en la séptima posición y $-(N_0, 7) = \emptyset$ es el conjunto de sucesiones en N_0 de la iteración 2, con 0 en la séptima posición.
$N_1 : +(N_0, 7) \cup N_1 = \emptyset$ este conjunto es la unión de las sucesiones en $+(N_0, 7)$, y N_1 el conjunto de la iteración anterior para $j = 1$
$N_0 : -(N_0, 7) = \emptyset$

Iteración 4

Para $i = 3$

$j = 3$ Separate $+(N_3, 8)$ y $-(N_3, 8)$
$+(N_3, 8)$=`{01111111,10111111,11101111}`, este conjunto son todas las sucesiones en N_3 de la iteración 3, con 1 en la octava posición y $-(N_3, 8)$=`{11011110}` es el conjunto de sucesiones en N_3 de la iteración 3, con 0 en la octava posición.
$N_4 : +(N_3, 8) \cup N_4$ =`{01111111,10111111, 11101111}` este conjunto es la unión de las sucesiones en $+(N_3, 8)$, y N_4
$N_0 : -(N_0, 7)$ =`{11011110}`

$j = 2$ Separate $+(N_2, 8)$ y $-(N_2, 8)$
$+(N_2, 8)=\emptyset$, este conjunto son todas las sucesiones en N_2 de la iteración 3, con 1 en la octava posición y $-(N_2, 8) = \emptyset$ es el conjunto de sucesiones en N_2 de la iteración 3, con 0 en la octava posición.

$N_3 : +(N_2, 8) \cup N_3$=$\{$`11011110`$\}$ este conjunto es la unión de las sucesiones en $+(N_2, 8)$, y N_3 el conjunto de la iteración anterior para $j = 3$
N_2:$-(N_2, 8)$=$\emptyset$

j=1 Separate $+(N_1, 8)$ y $-(N_1, 8)$
$+(N_1, 8)$=$\emptyset$, este conjunto son todas las sucesiones en N_1 de la iteración 3, con 1 en la octava posición y $-(N_1, 8)$=$\emptyset$ es el conjunto de sucesiones en N_1 de la iteración 3, con 0 en la octava posición.
N_2:$+(N_1, 8) \cup N_2$=$\emptyset$ este conjunto es la unión de las sucesiones en $+(N_1, 7)$, y N_2 de la iteración anterior para $j = 2$
$N_1 : -(N_1, 7) = \emptyset$

j=0 Separate $+(N_0, 8)$ y $-(N_0, 8)$
$+(N_0, 8)$=$\emptyset$, este conjunto son todas las sucesiones en N_0 de la iteración 3, con 1 en la octava posición y $-(N_0, 8) = \emptyset$ es el conjunto de sucesiones en N_0 de la iteración 3, con 0 en la octava posición.
$N_1 : +(N_0, 8) \cup N_1 = \emptyset$ este conjunto es la unión de las sucesiones en $+(N_0, 8)$, y N_1 el conjunto de la iteración anterior para $j = 1$
$N_0 : -(N_0, 8) = \emptyset$

Paso final El conjunto N_i, $0 \leq i \leq n$, contiene todas las memorias complejas en las cuales exactamente i de las subcadenas $m + 1$, ... ,$m + n$ son 1s. En este caso la iteración 4 nos dice que en el conjunto inicial N_0 de este algoritmo, existen 3 subgráficas de G con 4 vértices y una subgráficas con tres vértices y tres aristas (N_4=$\{$`01111111,10111111,11101111`$\}$ y N_3=$\{$`11011110`$\}$).

Algoritmo para el problema del Clan Máximo

Sea $1 \leq k \leq n$. Un clan de k vértices de una gráfica tiene $\binom{k}{2}$ aristas. La entrada de este algoritmo es $[m + n, \binom{m}{\binom{k}{2}}]$ biblioteca, y N_0 se encuentran codificados todos los $\binom{k}{2}$ subconjuntos de aristas. Este algoritmo determina cuáles de los subconjuntos de $\binom{k}{2}$ aristas forman un clan. Entonces el algoritmo funciona asi: primero determina las subgráficas de G inducidas por todos los subconjuntos de $\binom{k}{2}$ aristas por medio del algoritmo *Edge Induced Graphs*. Entre todas estas subgráficas se determinan cuales tienen k vértices, con el algoritmo *Weigthtening*. Estas subgráficas son exactamente los clanes con k vértices. En caso de que la gráfica G no contenga un clan de ese tamaño el algoritmo lo reporta. En el ejemplo del algoritmo anterior resulto una sucesión que representa la subgráfica K_3 de G

k`Cliques` (N_0, m, n, k)
$N_0 \leftarrow$ `Edge Induced Graphs` (N_0, m, n)

```
    N_0 ← Weigthtening (N_0, m, n, k)
    if N_0 is not empty then
        return N_0
    else
        report "no k-clique"
end kClique
```

Justificación del algoritmo

Sea $G = (V, A)$ una gráfica de orden n y tamaño m. El algoritmo comienza con todas la subgráficas inducidas por $\binom{k}{2}$ aristas de G, y en cada iteración se identifican que vértices de G forman parte de estas subgráficas. Separando las subgráficas en dos conjuntos; el primer conjunto contiene las subgráficas que tengan la i-ésima arista y el segundo las que no la tienen, después se van etiquetando los vértices extremos de la i-ésima arista. Esto se repite m veces. Al final de *Edge Induced graphs* se tienen todas la subgráficas inducidas por $\binom{k}{2}$ aristas de G y se sabrá qué vértices las forman. Para la segunda parte del algoritmo separa las subgráficas inducidas por $\binom{k}{2}$ aristas en dos conjuntos; uno que tenga las subgráficas que contengan el i-ésimo vértice y el otro las que no lo contienen. Así para cada vértice, y las va clasificando por el número de vértices, formando conjuntos de $k, k-1, k-2..,1$ vértices, en caso de no formar un conjunto con subgráficas de k vértices, quiere decir que no existe un clan de orden k en G. □

3.5. Simulación de membranas basado en ADN

En el 2007 Marc García-Arnau et al. [2] presentan una implementación de un P-sistema, por medio de un algoritmo basado en ADN, haciendo uso de estrategias constructivas, simulando el comportamiento de una estructura de membrana biológica. Debido a sus características, que actuan como barreras y filtros, separando la célula en distintas regiones y controlando el paso de moléculas entre las regiones. Esto inspiro para crear un sistema de membranas llamado *P-sistema* propuesto por Gheorghe Păun en 1998 [8]. Lo que caracteriza esta solución es el bajo costo en términos de tiempo, número y tamaño de las cadenas, así como la simplicidad de la implementación biológica.

P-sistema

La estructura de un P-sistema está delimitada por una *piel* que separa el sistema interno del ambiente exterior. En el interior de la piel hay un arreglo

jerárquico de membranas que definen regiones individuales. Se dice que una membrana es *elemental* cuando no contiene otras membranas y su región está definida por el espacio que encierra. Entonces una región definida por una membrana *no elemental* es el espacio entre está y la membrana contenida directamente dentro. Se le asigna un número natural a cada membrana, para hacerlas distinguibles durante el cómputo. Como cada región es delimitada por una única membrana, se usará el número de la membrana para referirnos a la región que delimita. Figura 3.7

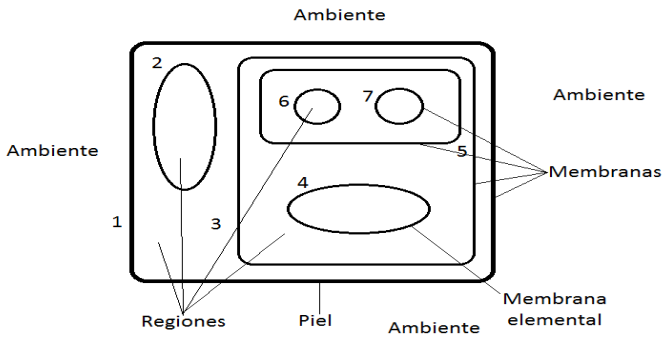

Figura 3.7: Estructura de membrana

Cada región contiene un conjunto de objetos y un conjunto de reglas. Los *objetos* son representados por símbolos de un alfabeto dado V. Las reglas transforman objetos y son de la forma *antes*→*después*, ésto significa que "cada instancia de antes se transforma a una instancia de después." Las reglas son representadas por un par (u, v) de sucesiones sobre el alfabeto V, con $u \in V$ y $v \in \{a_{here}, a_{out}\}$, donde a_{here} significa "una copia de a se queda en esta región", a_{out} significa "enviar una copia de a a través de la membrana y fuera de esta región". Las reglas de la forma $y \to x$ donde x y y son multiconjuntos de objetos, actúan de la siguiente manera. Por ejemplo si los objetos son a, b, c y están presentes en 5, 2 y 6 copias respectivamente, se puede representar este multiconjunto por la cadena $a^5b^2c^6$; todas las permutaciones de esta sucesión representan el mismo multiconjunto. Si se considera la siguiente regla $aab \to abcc$, ésto indica que cuando una sucesión contiene dos copias de a junto con una de b reacciona, y como resultado de esta reacción, se obtiene una copia de a con una copia de b y dos copias de c. Al aplicar esta regla al multiconjunto $a^5b^2c^6$ produce el multiconjunto $a^4b^2c^8$.

Un $P-$sistema con réplica reescrita e inhibidores para resolver PCM, formalmente es una construcción como sigue:

Sea una gráfica $G=(\{x_1, ..., x_n\}, E)$, donde los x_i son los vértices y E es el conjunto de arista, y $\bar{G} = (V, \bar{E})$ la gráfica complementaria.

Π =$(V,\ \mu, M_1,\ \ldots\ , M_n, R_1,\ \ldots\ , R_n)$ // Π *tiene una estructura de membrana* μ *compuesta de n membranas anidadas. Cada membrana contiene un conjunto de reglas* R_i *con replicación.*

V =$\{x_i, \mid 1 \leq i \leq n\ \} \cup \{d\}$, // *V el alfabeto de un P-sistema consiste de* $n+1$ *elementos: d un símbolo auxiliar, y* $x_1, ..., x_n$*, los vértices de la gráfica.*

M_1 =$\{d\}$, M_i=$\{\lambda\}$ con $2 \leq i \leq n$, // *Inicialmente contiene* $M_1 = d$ *y el resto de las membranas están vacias.*

R_i = $\{dw \to (dwx_i, out)_{\neg U_i}, dw \to (dw, out) \mid w \in \{x_j \mid 1 \leq j \leq i-1\}^*$, $0 \leq \mid w \mid \leq i-1\}$// *Cada regla es compuesta de un par de subreglas:* $r_{i,x}$*: dw* $\to$ *(dw*x_i*, out)*$_{\neg U_i}$ *esta regla a cada suseción w que no contenga elementos de* U_i *le agrega el dígito* x_i*, dando como resultado sucesiones del tipo* wx_i *y* x_i *que son las que pasarán a la siguiente membrana y* $r_{i,b}$*: dw* $\to$ *(dw, out) esta regla deja fijos los elementos w y hace que pasen a la siguiente membrana. El conjunto* A^* *denota el conjunto de todas las cadenas sobre el alfabeto A.*

U_i = $\{x_j \in V \mid \{x_i, x_j\} \in \bar{E}\}$, $1 \leq i \leq n$. // *También se define el conjunto de inhibidores* U_i*, conteniendo aquellos símbolos* a_j *que en presencia de éstos las reglas no pueden ser aplicadas*

El sistema comienza aplicando dos subreglas reescritas de la membrana 1 que inicialmente contiene una cadena representando el símbolo d ($M_1 = \{d\}$), mientras que las otras membranas permanecen vacías ($M_i = \{\lambda\}$ $2 \leq i \leq n$). Cuando una regla es aplicada a un objeto, éste primero es replicado en n copias y entonces cada subregla es aplicada a una copia. Al final del primer paso son creadas dos cadenas y enviadas a la siguiente membrana. Este proceso se repite en cada membrana aplicando sus respectivas reglas R_i, así en cada paso son generadas todas las subgráficas completas de la gráfica G. Nótese que las subreglas $r_{i,b}$ son siempre aplicables a todas las cadenas, mientras que las subreglas $r_{i,x}$ sólo son aplicables si las cadenas a ser reescritas que no contienen algún símbolo en el conjunto inhibidor U_i. En el paso n el lenguaje del sistema consiste de todos los clanes en la gráfica G y el más grande es la solución.
Para medir el tamaño del máximo clan, el P-sistema hace esta tarea con la estructura anteriormente definida Π con una pequeña variación que a continuación se presenta:

Π =$(V',\ \mu', M_1,\ \ldots\ , M_{n+1}, R'_1,\ \ldots\ , R'_{n+1})$

$V' = \{a_i, d_j \mid 1 \leq i \leq n,\ 0 \leq j \leq n\}$,

$\mu' = [_{n+1} \cdots [_2 [_1]_1]_2 \cdots]_{n+1}$ // *La estructura de membrana puede ser representada por diferentes expresiones de paréntesis.*

$M_1 = \{d_0\}$, $M_i = \{\lambda\}$ con $2 \leq i \leq n+1$,

$R'_i = \{d_k w \to (d_{k+1} w a_i, \text{out})_{\neg U_i},\ d_k w \to (d_k w, out) \mid$
$w \in \{a_j \mid 1 \leq j \leq i-1\}^*, \mid w \mid = k,\ 0 \leq k \leq i-1\}$
con $U_i = \{a_j \in V \mid \{a_i, a_j\} \in E'\}$, $1 \leq i \leq n$

$R'_{n+1} = \{\ d_k w \to (d_k w, here) \mid w \in \{a_j \mid 1 \leq j \leq n\}^*,$
$0 \leq \mid w \mid \leq n, 0 \leq k \leq n-1\}$
$\cup \{d_n w \to (d_n w, out) \mid w \in \{a_j \mid 1 \leq j \leq n\}^*,\ 0 \leq \mid w \mid \leq n\}$.

Este funciona de manera similar a Π con la única diferencia que en cada cadena que produce lleva información explícita, sobre el tamaño del clan que representa, codificado en el símbolo d_k. Después de n pasos las cadenas representando todos los clanes son reunidos en la membrana $n+1$. Y en el paso $n+1$ las cadenas representando clanes de tamaño n son enviadas fuera (si es que hay alguna), en el paso $n+2$ los clanes de tamaño $n-1$ y así sucesivamente.

Ejemplo:

Sea G la gráfica vista en la Figura 3.6

$\Pi = (V', \mu', M_1, M_2, M_3, M_4, M_5, R'_1, R'_2, R'_3, R'_4, R'_5\}$ donde
$V' = \{v_1, v_2, v_3, v_4\} \cup \{d_0, d_1, d_2, d_3, d_4\}$,
$\mu' = [_{n+1} \cdots [_2 [_1]_1]_2 \cdots]_{n+1}$

Iteración 1

$M_1 = d_0, M_i = \emptyset$ Al aplicar R'_1 a d_0 que es aplicar las subreglas:

$r_{1a} = d_0 w \to (d_1 w v_1, out)_{\neg U_1}$ con $U_1 = \{v_4\}$ y $w = \{\}$,
como $w \neq v_4$ se aplica la subregla r_{1a} y queda el conjunto $\{d_1 v_1\}$ que pasa a la siguiente membrana.

$r_{1b} = d_0 w \to (d_0 w, out)$ se aplica la subregla r_{1b} y queda el conjunto $\{\}$ que pasa a la siguiente membrana.

Iteración 2

$M_2 = \{d_1 v_1\}$ Al aplicar R'_2 a las sucesiones en M_2, que es aplicar las subreglas:

r_{2a} $=d_kw \to (d_{k+1}wv_2, out)_{\neg U_2}$ con $U_2 = \{v_4\}$ y $w = v_1, \mid w \mid= k, 0 \leq k \leq 1\}$
se aplica la subregla r_{2a} a todos los elementos en M_2 que no contienen a v_4 y queda el conjunto $\{d_2v_1v_2, d_1v_2\}$ que pasan a la siguiente membrana.

r_{2b} $=d_kw \to (d_kw, out)$ se aplica la subregla r_{2b} y queda el conjunto $\{d_1v_1\}$ que pasa a la siguiente membrana.

Iteración 3

$M_3 = \{d_1v_1, d_2v_1v_2, d_1v_2$ **}** Al aplicar R'_3 a las sucesiones en M_3, que es aplicar las subreglas:

r_{3a} $=d_kw \to (d_{k+1}wv_3, out)_{\neg U_3}$ con $U_3 = \{nulo\}$ y $w = v_1, v_1v_2, v_2 \mid w \mid= k$, $0 \leq k \leq 2\}$
se aplica la subregla r_{3a} a todos los elementos en M_3 que no contienen ningún vértice adyacente a v_3 en $\bar{G}$ y queda el conjunto $\{d_3v_1v_2v_3, d_2v_2v_3, d_2v_1v_3, d_1v_3\}$ que pasan a la siguiente membrana.

r_{3b} $=d_kw \to (d_kw, out)$ se aplica la subregla r_{3b} a todos los elementos en M_4 y queda el conjunto $\{d_1v_1, d_2v_1v_2, d_1v_2\}$ que pasa a la siguiente membrana.

Iteración 4

$M_4 = \{d_1v_1, d_2v_1v_2, d_1v_2, d_3v_1v_2v_3, d_2v_2v_3, d_2v_1v_3, d_1v_3$**}** Al aplicar R'_4 a las sucesiones en M_4, que es aplicar las subreglas:

r_{4a} $=d_kw \to (d_{k+1}wv_4, out)_{\neg U_4}$ con $U_4 = \{v_1, v_2\}$ y $w = v_3, \mid w \mid= k, 0 \leq k \leq 3\}$
se aplica la subregla r_{4a} a todos los elementos en M_4 que no contengan a v_1 y v_2 y queda el conjunto $\{d_2v_3v_4, d_1v_4\}$ que pasan a la siguiente membrana.

r_{4b} $=d_kw \to (d_kw, out)$ le aplico la subregla r_{4b} a todos los elementos en M_4 y queda el conjunto $\{d_1v_1, d_2v_1v_2, d_1v_2, d_3v_1v_2v_3, d_2v_2v_3, d_2v_1v_3, d_1v_3\}$ que pasa a la siguiente membrana.

Iteración 5

$M_5 = \{d_1v_1, d_2v_1v_2, d_1v_2, d_3v_1v_2v_3, d_2v_2v_3, d_2v_1v_3, d_1v_3, d_2v_3v_4, d_1v_4$**}** Al aplicar R'_5 a todas las sucesiones en M_5, que es aplicar las subreglas:

$d_kw \to (d_kw, here)$ con $0 \leq \mid w \mid \leq 4, 0 \leq k \leq 3 \cup d_4w \to (d_4w, out)$,
con $0 \leq \mid w \mid \leq 4$
La primera subregla es aplicada a sucesiones de tamaño menor o igual a tres y se quedan en la membrana M_5, la segunda regla es aplicada a las sucesiones de tamaño cuatro y las manda al exterior. Por lo tanto todas las sucesiones se quedan en M_5 indican que no existen clanes de tamaño cuatro.

Iteración 6

$M_5 = \{d_1v_1, d_2v_1v_2, d_1v_2, d_3v_1v_2v_3, d_2v_2v_3, d_2v_1v_3, d_1v_3, d_2v_3v_4, d_1v_4\}$ Al aplicar R'_5 a todas las sucesiones en M_5, que es aplicar las subreglas:

$d_kw \rightarrow (d_kw, here)$ con $0 \leq \mid w \mid \leq 4, 0 \leq k \leq 2 \cup d_3w \rightarrow (d_3w, out)$, con $0 \leq \mid w \mid \leq 3$

La primera subregla es aplicada a sucesiones de tamaño menor o igual a dos y se quedan en la membrana M_5, la segunda regla es aplicada a las sucesiones de tamaño tres y las manda al exterior. Por lo tanto se envían al exterior a la sucesión $v_1v_2v_3$ y el resto de las sucesiones se queda en M_5, entonces se tiene el máximo clan de tamaño tres en G.

Implementación

La implementación basada en la estructura y comportamiento de P-sistema con replica reescrita e inhibidores, se resume en el siguiente pseudocódigo. En este caso el algoritmo no utiliza cadenas de longitud constante, sino cadenas de tamaño variable sobre el alfabeto del problema. En este caso para PCM su alfabeto consta de los elementos del conjunto V de vértices de la gráfica. Por lo tanto un clan en la gráfica es codificado sólo por secuencias que representan a los vértices pertenecientes al clan, ya que los vértices que no pertenecen al clan son eliminados por las reglas.

A continuación se presenta la implementación que trabaja en tiempo lineal con respecto al número de vértices $\mid V \mid$ de la gráfica.

$\forall\ x_i \in V, i = 1, 2, ..., n.$ `begin`
{
`Replicate` (t_1, t_2, t_3)
$\forall\ (\{x_i, x_j\} \in \overline{E}$, con $1 \leq j \leq i-1)$`begin`
{
`Separate` (t_2, x_j, t_4)
`Delete` (t_4)
}
`Append` (t_2, x_i)
`Merge` (t_2, t_3, t_1)
}
`Measure strings` (t_1).

Justificación del algoritmo

Comienza con un conjunto vacío t_1, se irán construyendo todos los clanes posibles en G, con ayuda de la gráfica complementaria se sabrá qué conjunto de vértices descartar. Primero crear dos copias del conjunto vacío inicial en los conjuntos t_2 y t_3— y para todo x_j tal que $(x_i, x_j) \in A(\bar{G})$, extraer de t_2 los subconjuntos de vértices que tienen como elemento a los x_j y se ponen en el conjunto t_4 para luego ser eliminados. Después a los subconjuntos de t_2 que quedaron a cada uno agregar el elemento x_i. En el conjunto t_3 se guardan todas las permutaciones de elementos en t_2, por último se unen en el tubo t_1 y se cuenta el número de elementos en cada subconjunto de t_1. Se repite este proceso para cada vértice en G. Lo que resulta al final son todos los subconjuntos de vértices de G que forman un clan. Porque los que contienen elementos x_j tal que $(x_i, x_j) \in A(\bar{G})$ no pueden formar un clan en G y son eliminados todos al examinar esta propiedad para cada vértice de G. Como en cada iteración se va midiendo el tamaño de los clanes al final del algoritmo basta checar el máximo. □

Merge(t_1, t_2, t_3) Unir dos conjuntos diferentes de cadenas en tubos t_1 y t_2 a un tubo t_3.

Replicate(t_1, t_2, t_3): replica el contenido de un conjunto de cadenas de un tubo t_1 dentro de dos tubos t_2 y t_3 cada uno de estos contiene una copia del contenido original en t_1, el cual es vaciado. Esta función es implementada después de un ciclo de PCR.

Separate(t_1, a, t_2): Dado un tubo t_1 con cadenas y una subcadena a, se extraen todos las cadenas conteniendo como subcadena a, y se produce un tubo t_2 con todas estas cadenas extraidas.

Append(t_1, a): Dado un tubo t_1, agregar la cadena a a todos los finales de las cadenas contenidas en éste. En caso de que t_1 sea vacío el resultado de la operación será justo $\{\ a\}$.

Delete(t_1): Dado un tubo t_1, eliminar su contenido.

Measure strings(t_1): Mide las cadenas del tubo t_1 identificando la más grande. En su implementación se utiliza electroforesis en gel.

La implementación del P-sistema con ADN se basa en los siguientes puntos:

1. Las distintas membranas de un P sistema constituyen el espacio físico que delimita el alcance de réplica reescrita sobre las cadenas del problema en cada iteración. Por lo tanto el número de membranas está relacionado con el número de iteraciones necesarias para la implementación.

2. Como posee reglas R_i con réplica $(r_{i,x}, r_{i,b})$ el P-sistema es capaz de multiplicar su número de cadenas en cada paso por ello generando el incremento exponencial de espacio necesario para resolver problemas NP-completos en tiempo lineal. Desde el punto de vista de la implementación la propiedad de réplica es equivalente a la capacidad de dividir el espacio de las cadenas en varios subconjuntos, realizando diferentes operaciones en cada uno de ellos. La operación *Replicate* es la encargada de esta tarea. El número máximo de cadenas que pueden ser generadas después de que una regla es aplicada es llamado *grado de replicación*. Este grado de replicación se relaciona al número de subconjuntos (tubos) dentro del espacio de las cadenas de ADN que pueden ser replicadas. En este caso el grado de replicación del P-sistema es dos (cada regla R_i genera a lo más dos cadenas en cada paso), la operación Replicate ha de generar no más de dos subconjuntos t_2 y t_3 de un conjunto t_1.

3. La subregla $r_{i,x}$ de cada regla R_i del P-sistema, lleva acabo la operación reescribir que agrega un nuevo vértice x_i a las cadenas. Esta operación es realizada sólo en cadenas válidas; es decir, cadenas que no contienen el vértice a_j tal que $\{x_i, x_j\} \in E'$, con $i > j$. Para hacer ésto se requiere más de una operación, por ello la regla reescrita del P-sistema es relacionada a las operaciones Separate, Eliminate y Append.

4. Después de aplicar reglas del P-sistema a un conjunto de cadenas en una membrana en particular, las cadenas resultantes son enviadas al siguiente sistema de membrana, la cual es responsable de agrupar a todas las cadenas generadas por el último paso del sistema. Esta tarea es llevada a cabo por la operación *Merge* la cual debe ser capaz de agrupar tantos subconjuntos de cadenas que la operación *Replicate* genere.

5. Finalmente la operación *Measure strings*, determina la solución del problema.

Bibliografía

[1] L. Adleman, 1994. *Molecular computation of solutions to combinatorial problems.* Science 266, 1021-1024.

[2] Bäck, T., Kok, J.N., Rozenberg, G., 1999. *Evolutionary computation as a paradigm for DNA-based computing.* In: Landweber, L., Winfree, E., Lipton, R., Freeland, S. (Eds.). Proceedings of the DIMACS Workshop on Evolution as Computation. Princeton, NJ, pp. 67-88

[3] Head, T., Yamamura, M., Gal, S.,1999. *Aqueous computing: writing on molecules.*, In:Proceedings of Congress on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, pp. 1006-1010.

[4] Marc García-Arnau., Daniel, M., Alfonso Rodríguez-Patón., Petr, S., 2007 *A Psystem and a constructive membrane-inspired DNA algorithm for solving the Maximun Clique Problem.* Biosystems 90, 687-697.

[5] McCaskill, J.S., 2001. *Optically programming DNA computing in microflow reactors.* Biosystems 59, 125-138.

[6] Ouyang, Q., Kaplan, Peter, D., Liu, S., Libchaber, A., 1997. *DNA solution of the maximal clique problem.*, Science 278, 446-449.

[7] P. M. Pardalos and J. Xue., 1994. *The maximum clique problem.*, Journal of Global Optimization, 4:301-328.

[8] Păun, G., 2000. *Computing with membranes.*, J. Comput. Syst. Sci. 61, 108-143.

[9] Zimmermann, K.H., 2002. *Efficent DNA sticker algorithms for NP-complete graph problems.* Comput. Phys. Commun. 144, 297-309.